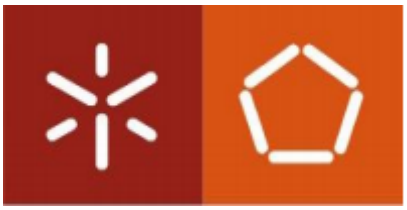

Universidade do Minho

Departamento de Sistemas de Informação

Manuel Alberto Araújo Martins da Silva

Aplicação do *Earned Value Management* na Metodologia *Scrum*

Desenvolvido sob a orientação do

Professor Doutor Pedro Abreu Ribeiro

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Outubro de 2017

Declaração RepositoriUM

Nome: Manuel Alberto Araújo Martins da Silva

Nº Cartão Cidadão /BI: 144620023 Tel./Telem.: 935155616

Correio electrónico: a64876@alunos.uminho.pt

Curso Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Ano de conclusão da dissertação: 2017

Área de Especialização: _____

Escola de Engenharia, Departamento/Centro: Departamento de Sistemas de Informação

TÍTULO DISSERTAÇÃO/TRABALHO DE *PROJECT*O:

Título em PT : Aplicação do *Earned Value Management* em modelos ágeis

Título em EN :

Orientador Professor Doutor Pedro Abreu Ribeiro

Co-orientador_____

Nº ECTS da Dissertação_____ Classificação em valores (0-20)_____

Classificação ECTS com base no percentil (A a F)_____

Declaro sob compromisso de honra que a dissertação/trabalho de *project*o agora entregue corresponde à que foi aprovada pelo júri constituído pela Universidade do Minho.

Declaro que concedo à Universidade do Minho e aos seus agentes uma licença não-exclusiva para arquivar e tornar acessível, nomeadamente através do seu repositório institucional, nas condições abaixo indicadas, a minha dissertação/trabalho de *project*o, em suporte digital.

Concordo que a minha dissertação/trabalho de projeto seja colocada no repositório da Universidade do Minho com o seguinte estatuto (assinale um):

3.4.1 ☐ Disponibilização imediata do trabalho para acesso universal;

3.4.2 ☐ Disponibilização do trabalho para acesso exclusivo na Universidade do Minho durante o período de ☐ 1 ano, ☐ 2 anos ou ☐ 3 anos, sendo que após o tempo assinalado autorizo o acesso universal.

3.4.3 ☐ Disponibilização do trabalho de acordo com o **Despacho RT-98/2010 c**) (embargo_____ anos)

Braga/Guimarães, _____ /_____ /_____

Assinatura: _____

Agradecimentos

A realização desta Dissertação de mestrado contou com importantes apoios e incentivos sem os quais não se teria tornado uma realidade e aos quais estarei eternamente grato.

Quero agradecer aos meus pais pela confiança depositada em mim desde sempre e por todas as oportunidades fantásticas que me proporcionaram ao longo da vida e por todo o conhecimento que me forneceram e fizeram de mim a pessoa que sou hoje e que me permitiu realizar esta dissertação e concluir o Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação, sem eles não seria possível.

Um agradecimento especial ao meu irmão pelo espírito crítico e por todo o conhecimento que me passou e por todas as dificuldades que me ajudou a superar. Obrigado pela confiança e motivação que depositaste em mim.

Um agradecimento enorme ao professor Pedro Ribeiro pela disponibilidade que me ofereceu, pelo conhecimento e pelos conselhos que me facultou e pela confiança depositada em mim ao aceitar ser meu orientador na Dissertação.

Não poderia ainda deixar de agradecer aos meus colegas e amigos que sempre me ofereceram motivação e encorajaram a dar o meu melhor e que me aturaram ao longo deste longo percurso.

Por último não poderia deixar de agradecer à minha namorada por todo o apoio e carinho que me deu.

Resumo

Pretende-se com este trabalho avaliar a aplicação do método EVM no contexto de projetos de desenvolvimento de *software*, utilizando a metodologia ágil *Scrum*.

Este projeto de dissertação tem o objetivo de estudar as propostas mais recentes do método EVM em relação à monitorização dos prazos de um projeto, definir e configurar um ambiente adequado para a aplicação do método EVM em projetos ágeis e fazer uma proposta de um referencial rigoroso para aplicação do método EVM na metodologia ágil *Scrum*.

Os métodos ágeis utilizam planeamento estruturado para desenvolvimento de *software*, com um envolvimento contínuo entre o cliente e a equipa de desenvolvimento, com entregas frequentes de *software* funcional.

Foi realizado um estudo extensivo tanto do método EVM como dos modelos ágeis. Foi também estudado como proceder á aplicação do método EVM em *Scrum* de forma a que as vantagens trazidas do método EVM para os modelos ágeis sobreponham a desvantagem de tornar os modelos ágeis, não tão ágeis.

Neste projeto de dissertação propõe-se a realização de uma proposta de um referencial rigoroso conjugando as várias vantagens da aplicação do método EVM com a abordagem ágil *Scrum* de modo a criar um referencial com uma margem de sucesso satisfatória em projetos de *software* utilizando a metodologia *Design* Science in Information Systems *Research*.

Abstract

This work intends to evaluate the application of the EVM method in the context of software development projects, using Scrum agile methodology

This dissertation project aims to study the most recent proposals of the EVM method in relation to the monitoring of the deadlines of a project, to define and configure an adequate environment for the application of the EVM method in agile projects and to propose a rigorous reference for Application of the EVM method in Scrum agile approach.

Agile methods use structured planning for software development, with continuous engagement between the client and the development team with frequent deliveries of functional software.

To date, an extensive study of both the EVM method and the agile models has been done. It was also studied how to apply the EVM method in Scrum so that the advantages brought by the EVM method for agile models overlap the disadvantage of making models agile, not so agile.

In this dissertation project it is proposed to make a rigorous reference proposal combining the various advantages of applying the EVM method with the Scrum agile approach in order to create an artefact with a satisfactory margin of success in software projects using the Methodology Design Science in Information Systems Research.

Índice

Índice de Figuras

Índice de Tabelas

Siglas e Acrónimos

AC - custo atual

ACP - certificado pr*of*issional de ágil

BAC - orçamento na conclusão de projeto

BDD – desenvolvimento orientado para comportamento

BCWP - custo orçamentado de trabalho realizado

BCWS - custo orçamentado de trabalho programado

CFD – *diagrama* de fluxo cumulativo

CMM - modelo de maturidade de capacidade

CPM – método do caminho crítico

CPO - índice de desempenho de custo

CV - variação de custo

DSDM - método de desenvolvimento de sistemas dinâmicos

EAC - estimativa de conclusão

ETC - estimativa para acabar

EV - valor ganho

EVM - *earned value management*

FDD - desenvolvimento orientado para recursos

PERT – técnica de avaliação e revisão de programas

PM - gestão de projetos

PMBOK - *project management* body *of knowledge*

PMI - *project management* institute

PV - valor planeado

SDLC - ciclo de vida de desenvolvimento de *software*

SI - sistemas de informação

SPI - índice de desempenho da programação

SV - variação da programação

SWEBOK - *software engineering* body *of knowledge*

ROI - retorno sobre investimento

TDD – desenvolvimento orientado para testes

WBS – *work* breakdown structure

XP - programação extrema

1. Introdução

Neste capítulo é feito o enquadramento e apresentada a estrutura do projeto de dissertação.

1.1 Enquadramento

Este tema de dissertação de mestrado está inserido na área de gestão de projetos, mais concretamente na gestão de projetos de *software*, sendo esta uma das áreas cruciais e com mais impacto no domínio dos Sistemas de Informação.

A gestão de projetos é a aplicação de conhecimento, ferramentas, técnicas e competências para as atividades irem de encontro aos requisitos do projeto (PMI, 2013).

A gestão de projetos é uma das áreas de foco das tecnologias e sistemas de informação, sendo que é uma atividade que pode e deve ser aplicada nos projetos de desenvolvimento de aplicações informáticas. Estas atividades em conjunto com outros métodos e metodologias aumentam a probabilidade de sucesso final do projeto.

A gestão de projetos engloba cinco grupos de processos: iniciação, planeamento, execução, monitorização e controlo e encerramento do projeto. Este projeto de dissertação irá essencialmente focar-se no planeamento e na monitorização e controlo.

É no âmbito da gestão de projetos que surge o método EVM, introduzido no guia PMBOK, que tem um papel fundamental na ajuda aos gestores de projeto, na medição do desempenho de projetos (Gonçalves, C. 2016). Existem as abordagens tradicionais onde o EVM está inserido assim como as abordagens ágeis, menos convencionais e mais flexíveis.

Segundo o *Chaos report* realizado pelo *Standish Group* (Hastie & Wojewoda, 2015), num estudo realizado entre 2011 até 2015 com uma amostra de 50.000 projetos de desenvolvimento de *software* apenas 29% em média durante os 5 anos de estudo têm uma classificação de sucesso de acordo com 6 métricas de sucesso: tempo, orçamento, alvo, objetivo, valor, satisfação. Este estudo fez ainda uma comparação entre a metodologia *waterfall* com a metodologia ágil, sendo que a metodologia ágil teve uma percentagem de sucesso superior à metodologia *waterfall* em todas as escalas de projeto, desde projetos de pequeno tamanho a grandes projetos (ver figura 1)Figura 1 - Ágil vs Waterfall (fonte: https://www.infoq.com/articles/standish-chaos-2015).

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%

Figura 1 - Ágil vs Waterfall (fonte: https://www.infoq.com/articles/standish-chaos-2015)

Atualmente ainda existem pessoas que pensam que as metodologias ágeis são apenas utilizadas em projetos de *software* de pequena escala e que as metodologias mais tradicionais que utilizam o EVM são utilizadas para projetos mais complexos, mas essa realidade está a mudar como comprova o Instituto VersionOne com os seus *surveys* sobre metodologias ágeis (VersionOne, 2006).

As metodologias ágeis já não são apenas o domínio de *startups* e pequenas lojas de desenvolvimento. No estudo do *Standish Group* (Hastie & Wojewoda, 2015) que ocorreu entre 2006 e 2015, foram entrevistados colaboradores de organizações de desenvolvimento de software ágil. Em 2006, quase dois terços dos entrevistados disseram que trabalham em organizações com menos de 100 pessoas. Até 2015, quase dois terços dos entrevistados disseram trabalhar para organizações com mais de 100 pessoas, e 31% disseram ter trabalhado para organizações com mais de 1.000 pessoas.

Earned Value Management tem sido estabelecida como uma das ferramentas de gestão de projetos mais eficazes na gestão de âmbito, cronograma e custo, com as suas raízes remontando ao final dos anos 1800 (Su-Cheng Wu, 2013).

Segundo o PMI (2006), EVM é um método que combina âmbito, cronograma e dados de recursos numa medida de desempenho e progresso, comparando o que foi orçamentado para uma tarefa (tempo e recursos) contra o que a tarefa realmente exigiu (tempo e recursos).

O movimento ágil agora está espalhado por todo o mundo. Tem sido introduzido em todos os tipos de indústrias com alta popularidade entre as várias equipas de desenvolvimento. Adotar métodos ágeis significa incorporar novas abordagens dinâmicas em termos de papéis, estilo de gestão, monitorização e controlo de projetos de desenvolvimento de *software* (Kane, 2007).

As empresas estão a mover-se para os métodos ágeis porque o mercado exige mais flexibilidade, mais rápida reatividade com expectativas de alta qualidade. Os métodos ágeis prometem às empresas uma entrega mais rápida e menos despesas para operações no seu desenvolvimento de *software* (Kane, 2007).

É corrente encontrar menção à recusa de incorporação do EVM em metodologias ágeis devido aos requisitos inicias de software não serem um dado adquirido nas metodologias ágeis, mas já existem estudos (Cabri & Griffiths 2006; Sulaiman, Barton, & Blackburn, 2006) que mostram que é possível e útil essa incorporação, com certas adaptações.

Assim, neste projeto de dissertação propõe-se a realização de um modelo de gestão de projetos que combine tanto a técnica EVM como a modelação ágil *Scrum*.

1.2 Finalidade e principais objetivos

Pretende-se com este trabalho estudar a aplicação do método EVM (e evoluções mais recentes) no contexto de projetos de desenvolvimento de *software*, utilizando a metodologia ágil *Scrum*. Espera-se uma definição e modelação da implementação do método EVM em *Scrum*. Nesta perspetiva os objetivos principais são:

- Estudo do método EVM, incluindo as propostas mais recentes em relação à monitorização dos prazos de um projeto;
- Definição e configuração de um ambiente adequado para a aplicação do método EVM em *Scrum*;
- Com base nos casos práticos, proposta de um referencial rigoroso para aplicação do método EVM em abordagens ágeis;

1.3 Descrição e organização do documento

No primeiro capítulo, a Introdução descreve o enquadramento do trabalho, apresentando a finalidade, definindo os objetivos principais da dissertação, descreve de forma sintetizada a estrutura do documento.

No segundo capítulo, o Enquadramento Conceptual, encontra-se definida a estratégia de pesquisa utilizada e a abordagem metodológica.

No terceiro capítulo, a Revisão de Literatura, é realizada a descrição dos vários conceitos essenciais abordados por este projeto de dissertação assim como a análise dos estudos que caracterizam o estado da arte relativamente às metodologias objeto de estudo nesta dissertação.

No quarto capítulo, o *Scrum* e o porquê da sua utilização nesta dissertação, aparece um detalhe mais rigoroso de como aplicar a metodologia e quais os papéis existentes, os eventos e artefactos.

No quinto capítulo, são apresentadas as contribuições deste projeto de dissertação assim como o referencial para a aplicação do método EVM na metodologia ágil *Scrum*.

No sexto capítulo, a Conclusão, encontra-se a conclusão e as propostas de trabalhos futuros.

No último capítulo, as Referências, é onde se encontram todo o repositório científico auxiliar para a realização deste projeto de Dissertação.

2. Metodologia de Investigação

Neste capítulo fica descrita a estratégia de pesquisa utilizada na revisão de literatura e a apresentação da abordagem metodológica adotada no desenvolvimento da dissertação.

2.1 Abordagem metodológica

No desenvolvimento deste trabalho de investigação será seguida a metodologia *Design* Science in Information Systems *Research* (Hevner, March, Park, & Ram, 2004). Esta metodologia engloba um conjunto de técnicas e perspetivas analíticas para realizar investigação com rigor em Sistemas de Informação.

As orientações desta metodologia são as seguintes:

- *Design as an artifact*

Deve produzir um artefacto viável sob a forma de uma construção de um modelo, um método, ou uma instanciação.

- *Problem* relevance

Desenvolver soluções com base em tecnologia para problemas importantes e relevantes de negócio.

- *Design evaluation*

A utilidade, qualidade e eficácia de um artefacto de *design* deve ser rigorosamente demonstrada através de métodos de avaliação bem executados.

- *Research contributions*

Deve fornecer contribuições claras e verificáveis nas áreas do *design* do artefacto, do *design* de fundações e / ou metodologias de projeto.

- *Research rigor*

Baseia-se na aplicação de métodos rigorosos, tanto na construção como na avaliação do *design* do artefacto.

- *Design as a search process*

Requer a utilização de meios disponíveis para atingir os fins desejados, desde que satisfaçam as leis no ambiente do problema.

- *Communication of research*

Deve ser apresentado de forma eficaz tanto para o público da área da gestão, bem como o público da área da tecnologia.

2.2 Pesquisa de Literatura Relevante

Nesta secção apresenta-se a estratégia de pesquisa, as fontes de dados e a seleção dos artigos.

2.2.1 Estratégia de Pesquisa e Fonte dos Dados

Para a revisão de literatura foi efetuado uma pesquisa extensiva em diversas bases de dados relevantes, tendo em conta a sua credibilidade onde a qualidade da informação é assegurada através de padrões mínimos reconhecidos. As seguintes bases de dados foram utilizadas para a realização da revisão de literatura: Bases de dados referencias, Bases de dados com texto integral, coleções de e-Revistas e repositórios científicos. Na tabela 1 estão listadas as bases de dados pesquisadas, a expressão utilizada e os artigos utilizados bem como o total de artigos obtidos.

Tabela 1 – Detalhes sobre a pesquisa bibliográfica realizada

Bases de Dados	Expressão	Utilizados/Total
Web *of* Science	Evm AND *agile*	3/13
SCOPUS	Evm AND *agile*	4/19
Academic Search Complete	Evm AND *agile*	1/4
ScienceDirect	Evm AND *agile*	1/30
Springer	Evm AND *agile*	1/38
Wiley InterScience	Evm AND *agile*	1/34
Google Scholar	Evm AND *agile*	11/1320
BASE	Evm AND *agile*	2/5
OpenAIRE	Evm AND *agile*	1/2
RCAAP	Evm AND *agile*	1/2
IEEE database	Evm AND *agile*	5/39

2.2.2 Seleção dos artigos

Foram selecionados artigos científicos de conferência ou periódicos, teses de mestrado e livros reconhecidos e referenciados na área de sistemas de informação.

Numa primeira fase, todos os artigos disponíveis com as palavras-chave EVM e *agile* foram recolhidos e estudados devido à escassez de informação, sendo este assunto relativamente recente e pouco explorado. Muitos dos artigos utilizavam as duas palavras-chave, mas a informação era sobre a área digital das telecomunicações, logo eram excluídos através de uma breve leitura do *abstract*.

Para que os artigos fossem aceites para a revisão de literatura tinham que responder a pelo menos um de três critérios:

- Abordar o método EVM;
- Abordar metodologias ágeis;

- Abordar o método EVM e as metodologias ágeis.

Para completar a revisão de literatura foram adicionados livros reconhecidos na área de sistemas de informação. Estes livros abordavam áreas tanto de gestão de projeto como de engenharia de *software* (SWEBOK, PMBOK, PRINCE2, etc.). Estes livros foram muito importantes para fundamentar as bases do estudo deste projeto de dissertação assim como assimilar conceitos necessários para o estudo, literatura e trabalho futuro.

2.2.3 Extração dos Dados

Neste ponto encontra-se o cruzamento das referências com os conceitos fundamentais deste projeto de dissertação.

Tabela 2 - Mapa de conceitos

Referências/Conceitos	EVM	*Agile*	PM	SI
(Anbari, 2003)	X		X	
(Bassil, 2012)				X
(Beck, 2001)		X	X	
(Beck, 1999)		X	X	
(BURBA, 2015)	X	X	X	
(Cabri & Griffiths, 2006)	X	X	X	
(Downey & Sutherland, 2013)		X		
(C. Gonçalves, 2016)	X	X	X	X
(Fowler & Highsmith, 2001)		X	X	
(Glaiel, 2012)	X	X	X	X
(Hall, 2012)	X	X	X	
(Hartmann & Dymond, 2006)		X		
(Hayes, Miller, Lapham, Wrubel, & Chick, 2014)	X	X	X	X
(Hevner et al., 2004)				X
(Highsmith J., 2002)		X	X	
(Highsmith, 2002)		X	X	
(Ian Sommerville, 2010)	X	X	X	X
(Ilieva, Ivanov, & Stefanova, 2004)		X	X	
(F. Gonçalves, 2013)	X	X	X	
(Kane, 2007)	X	X	X	X
(Kantor et al., 2016)	X	X	X	

(Kerzner, 2009)	X		X	X
(Mahalakshmi & Sundararajan, 2013)		X	X	
(Manual, 2006)	X	X	X	
(Munawar & M., 2015)		X	X	
(Palmquist, Lapham, Miller, Chick, & Ozkaya, 2013)	X	X	X	
(Park, 2010)	X	X		
(PMI, 2011)	X		X	X
(PMI, 2013)	X	X	X	X
(PMI, 2006)	X		X	X
(Pressman, 2009)		X	X	X
(Roy & Goutam, 2014)	X	X		
(Rusk, 2009)	X	X	X	
(Seetharaman & Mansor, 2015)	X	X	X	X
(Solomon, 2005)	X		X	
(Sulaiman et al., 2006)	X	X	X	
(Torrecilla-Salinas, Sedeño, Escalona, & Mejías, 2015)	X	X	X	X
(Turley & *Of*fice *of* Government Commerce, 2010)		X	X	
(Wu, 2012)	X	X	X	
(Vani, Suriya, & Deepalakshmi, 2014)	X	X	X	
(VersionOne, 2006)		X		
("2010_State_*of_Agile*_Development_*Survey*_Results.pdf," n.d.)		X		
(Abran, Moore, Dupuis, & Tripp, 2004)		X	X	X
(Cohen & Gan, 2014)	X	X	X	

3 Revisão de Literatura

Neste capítulo são abordados os temas chave deste projeto assim como as duas temáticas as metodologias tradicionais e as ágeis.

3.1 Engenharia de *Software*

A engenharia de *software* é cada vez mais reconhecida como um assunto importante e digno de investigação. Por toda a indústria o engenheiro de *software* substituiu o programador como o título de preferencial de trabalho. Modelos de processo de *software*, métodos de engenharia de *software* e ferramentas de *software* foram adotados com sucesso num amplo espectro de aplicações da indústria (Pressman, 2009).

A engenharia de *software* é uma disciplina de engenharia que engloba todos os aspetos da produção de *software*, desde o planeamento do projeto até à entrega do *software* (Ian Sommerville, 2010).

O objetivo principal do desenvolvimento de *software* é fornecer aos clientes produtos de qualidade sem defeitos e que cumpram todas as expectativas dos utilizadores finais. Para atingir esses objetivos, várias metodologias de Ciclo de Vida de Desenvolvimento de *Software* (SDLC) foram usadas (Seetharaman & Mansor, 2015).

Alguns dos métodos de desenvolvimento tornaram-se fortemente orientados para a documentação ou esperam que os desenvolvedores sigam rigorosamente certos processos. Esses podem ser chamados de métodos pesados ou tradicionais, por exemplo a metodologia *Waterfall* (Seetharaman & Mansor, 2015).

Engenharia de *software* continua a ser uma espécie de junção das novas tecnologias com as práticas recentes. A aceitação de novas técnicas cresce e as técnicas mais antigas são descartadas (Abran et al., 2004).

3.2 Gestão de Projetos

A gestão de projetos evoluiu para um processo de negócio e não para apenas um processo que faz a gestão de projetos. Cada vez mais empresas estão agora a ver a gestão de projetos como sendo obrigatória para a sobrevivência da empresa (Kerzner, 2009).

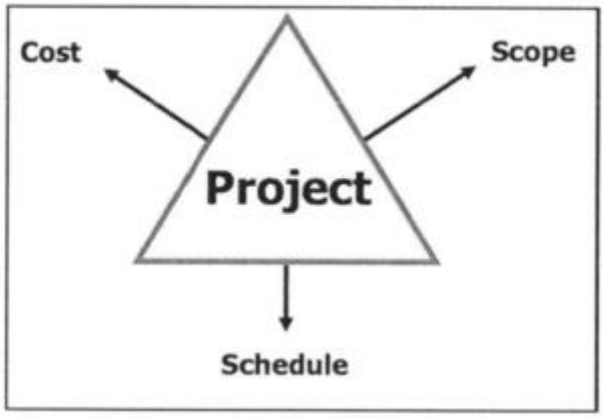

Figura 2 – Iron Triangle (Glaiel, 2012)

Estas três restrições são referidas como o "Triângulo de Ferro" Figura 2 – Iron Triangle *(Glaiel, 2012)*da gestão de projetos figura 2. Este é um paradigma útil para o medir o desempenho do projeto. Num mundo ideal, um projeto de *software* é bem-sucedido quando oferece o conjunto total de funcionalidades, dentro do prazo conforme planeado e dentro do orçamento. Os gestores de projeto para entregar um *software* bem-sucedido devem ajustar os objetivos do projeto, escolhendo quais os lados das restrições a relaxar. Quando um projeto encontra problemas de desempenho, as opções do gestor são puxar uma das três alavancas do triângulo de ferro:

- Aumentar o esforço (e, portanto, o custo), autorizando horas extra ou contratando mais funcionários;
- Relaxar o horário, atrasando tarefas ou a entrega do *software*;
- Cortar o âmbito, adiando um subconjunto de recursos para futuras versões de *software* ou reduzindo a qualidade dos recursos fornecidos.

Segundo Hall (2012), o processo de negócio conhecido como gestão de projetos tem tido um crescimento notável nos últimos 15 anos, como demonstrado por um aumento de 1000% na adesão ao *Project Management Institute* desde 1996. Este crescimento é em grande parte atribuível ao surgimento de muitas novas aplicações de diversos negócios que podem ser geridos com sucesso como projetos. As novas aplicações para gestão de projetos incluem implementações de TI, pesquisa e desenvolvimento, desenvolvimento de novos produtos e serviços, gestão de mudanças corporativas e desenvolvimento de *software* (Hall, 2012).

A gestão de projetos é necessário para planear melhor, monitorar o trabalho, fazer numerosos controlos e entregas, lidar com os riscos, lidar com as questões que surgem e identificar áreas para cortar custos (Turley, 2010).

Os processos de estimativa, planeamento e gestão são cruciais para projetos de desenvolvimento de *software*, uma vez que os resultados devem estar relacionados a várias estratégias de negócios (Torrecilla-Salinas et al., 2015).

A história da metodologia de gestão de projetos é vasta, desde CPM (método do caminho crítico) e PERT (técnica de avaliação e revisão de programas) até às influentes direções modernas de gestão de projetos de cadeia crítica e métodos ágeis (Hall, 2012).

As organizações que eram oponentes da gestão de projetos são agora obsoletas (Kerzner, 2009).

3.3 Waterfall

O modelo *waterfall* é um modelo de processo de desenvolvimento de *software* sequencial que funcionam, através de uma lista de fases que devem ser executadas para construir com sucesso um projeto de desenvolvimento de *software*.

As várias fases do modelo *waterfall* encontram-se identificadas na figura 3Figura 3 - Waterfall model (Bassil, 2012).

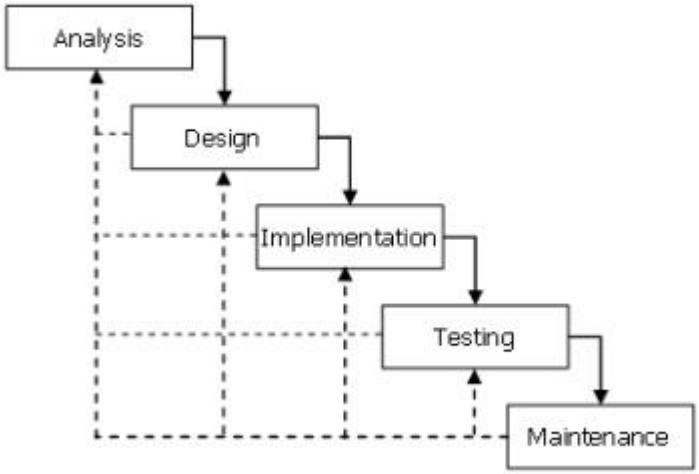

Figura 3 - Waterfall model (Bassil, 2012)

Essencialmente, o modelo *Waterfall* compreende cinco fases: análise, conceção, implementação, teste e manutenção.

Segundo Bassil (2012), as fases do modelo *waterfall* são:

Fase de Análise: geralmente conhecida como Especificação de Requisitos de *Software* (SRS - *Software* Requirements Specification) é uma descrição completa e abrangente do comportamento do *software* a ser desenvolvido. Implica que analistas de sistemas e de negócios definam requisitos funcionais e não funcionais.

Fase de Conceção: é o processo de planeamento e definição de uma solução para o problema, por forma a cumprir as expectativas dos *stakeholders*. Implica a contribuição de arquitetos de *software* e *designers* para conceber uma solução.

Fase de Implementação: refere-se à realização de requisitos de negócio e especificações de projeto num programa concreto e executável, bases de dados, site ou componentes de *software* através da programação e transição.

Fase de teste: é também conhecida como verificação e validação, que é um processo para verificar se uma solução de *software* atende aos requisitos e especificações originais e se cumpre os objetivos propostos.

Fase de manutenção: é o processo de modificar uma solução de *software* após a entrega e a transição para refinar outputs, corrigir erros e melhorar o desempenho e a qualidade.

Os requisitos são reunidos na etapa inicial. As mudanças de requisitos no âmbito são de difícil incorporação no método *waterfall*, uma vez que normalmente implica retrabalho em algumas fases iniciais do processo. Os *milestones* e a estrutura de divisão de trabalho (WBS) para toda a aplicação de *software* são decididos no início do projeto, durante o processo de planeamento. A abordagem *waterfall* torna-se mais cara se uma mudança acontece em fases posteriores (Roy & Goutam, 2014).

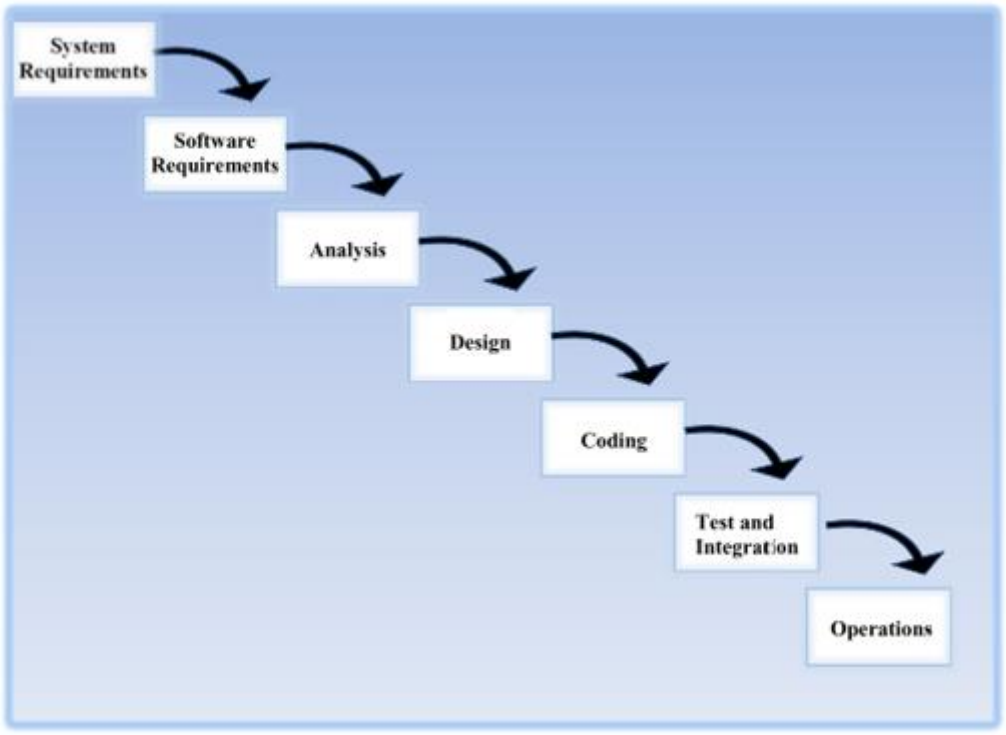

Figura 4 - Representação básica do modelo waterfall ((Palmquist et al., 2013)

A partir da representação básica do modelo (Figura 4), podemos ver os principais pontos que com ou sem razão vieram a ser os pilares da abordagem tradicional de desenvolvimento de *software waterfall* (Palmquist et al., 2013):

- Os sistemas são desenvolvidos num processo sequencial;
- Todos os requisitos são determinados no início do projeto, com os dois pressupostos de que todos os requisitos podem ser conhecidos no início do projeto e que continuarão inalterados;
- A análise é feita uma vez e precede o *design*;
- O *design* é feito uma vez e precede a codificação;
- Toda a codificação é feita uma vez e precede o teste e a integração;
- Todos os testes são feitos uma vez ou em conjuntos seguidos de atividades de integração, e precedem o uso operacional.

De seguida seguem mais alguns pontos utilizados na metodologia *waterfall*:

- É necessário uma revisão e aprovação formal para avançar de uma etapa para a seguinte;
- O cliente está mais envolvido na configuração dos requisitos, a maioria não se envolve durante as fases de *design*, de análise e a codificação e reaparecem durante o teste e a aceitação;
- É necessária uma extensa documentação para cada etapa.

Havia uma série de razões pelas quais essa abordagem parecia boa, e não podemos exagerar que essa abordagem era uma resposta ao risco e à incerteza. O modelo *waterfall* foi visto como uma técnica de gestão de riscos e nasceu num momento em que o próprio desenvolvimento do sistema estava centrado em hardware e aconteceu a um ritmo muito mais lento do que o que vemos hoje (Palmquist et al., 2013).

Alguns dos principais pontos de gestão de risco e os raciocínios que foram incorporados à metodologia tradicional *waterfall*, podem ser resumidos nos seguintes (Palmquist et al., 2013):

- Identificar precocemente todos os requisitos para apoiar o planeamento e a orçamentação;
- Identificar e bloquear os requisitos no início do programa para evitar problemas de âmbito;
- Documentar todos os aspetos do *design* e da tomada de decisão é necessário para futuras referências;
- Extensas revisões não só garantem que todos os *stakeholders* estão conscientes do progresso, mas também permitem que as questões sejam descobertas com mais antecedência;
- É necessário que todos os requisitos estejam documentados antes de qualquer trabalho de *design* para garantir que a arquitetura é adequada;

- Todas as codificações necessitam de estar concluídas antes do teste para garantir que todas as capacidades do sistema possam ser avaliadas;
- Todos os requisitos devem ser cumpridos antes que o sistema possa ser entregue.

3.4 *Earned Value Management*

O *Earned Value Management* é uma técnica de gestão de projetos que mede, numa data específica, o progresso e desempenho de um projeto em relação ao plano estipulado e estima o desempenho futuro (Cabri & Griffiths, 2006).

A metodologia EVM aborda três áreas do PMBOk - o âmbito, o prazo e os custos do projeto, e aplica-se a muitas das Áreas de Conhecimento e Grupos de Processo do Guia PMBOK (PMI, 2011).

Segundo Wu (2012) *Earned Value Management* (EVM) tem sido estabelecida como uma das ferramentas de gestão de projetos mais eficazes na gestão do âmbito, tempo e custo, com as suas raízes remontando ao final dos anos 1800. No entanto, estas ferramentas altamente valiosas não foram aplicadas ao *framework* cada vez mais popular da gestão de projetos ágeis (Wu, 2012).

A gestão de projetos ágeis é um conceito de gestão moderno desenvolvido para lidar com projetos altamente dinâmicos que têm requisitos e prazos de entrega que são difíceis de definir nos estados iniciais do projeto. Como o âmbito desses projetos muda constantemente, as técnicas EVM tradicionais são de aplicação difícil ou limitada (Wu, 2012).

Os dados do EVM serão fiáveis e precisos somente se forem selecionadas as medidas corretas de desempenho técnico e se forem avaliadas objetivamente (Solomon, 2005).

EVM é bem conhecido como sendo utlizado para o método *waterfall*. É amplamente conhecido o rastreamento do custo e âmbito para a execução do projeto. Normalmente existe o pressuposto de que EVM não pode ser implementado com os métodos ágeis. Os métodos ágeis são baseados em *sprints*, enquanto o método *waterfall* é baseada em *milestones*. A aplicação tradicional de *software* é amplamente orientada para seguir o desenvolvimento sequencial do ciclo de vida de *software* (Roy & Goutam, 2014).

Ao mesmo tempo, as agências de financiamento prescrevem a contabilidade de gestão de *Earned Value* para grandes projetos que inevitavelmente incluem componentes substanciais de *software*. As abordagens de *Earned Value* dão enfâse a um plano mais abrangente e tipicamente de longo alcance, e tendem a caracterizar novos planos frequentes e novas priorizações como indicativos de problemas (Kantor et al., 2016).

Earned Value é uma técnica de monitorização e relatório de gestão de projetos que foi desenvolvida e utilizada ao longo dos últimos 100 anos em projetos de engenharia tradicional. Baseia-se num plano inicial

com base em tarefas para medir o progresso de um projeto com um âmbito bem definido que evolui de forma sequencial e linear (Cabri & Griffiths, 2006).

Nas tabelas 3 e 4 apresentam-se os principais conceitos e cálculos utilizados pelo EVM.

Tabela 3 - Elementos de dados chave de EVM (Glaiel, 2012)

Questão	Elementos de dados
Quanto trabalho a ser feito?	*Budgeted Cost for Work Scheduled* (BCWS)
Quanto trabalho feito?	*Budgeted Cost for Work* Performed (BCWP)
Quanto custa o trabalho feito?	Actual *Cost of Work* Performed (ACWP)
Qual o custo total do trabalho?	*Budget at Completion* (BAC)
O que se espera que custe o trabalho total?	*Estimate at Completion* (EAC)

Tabela 4 - Principais cálculos da análise de dados EVM (Glaiel, 2012)

Métrica	Símbolo	Fórmula
Percentagem concluída	% Feito	$\frac{BCWP}{BAC}$
Índice de desempenho do Custo	CPI	$\frac{BCWP}{ACWP}$
Índice de desempenho de trabalho a completar	TCPI	$\frac{BAC - BCWP}{EAC - ACWP}$
Índice de desempenho do cronograma	SPI	$\frac{BCWP}{BCWS}$
Estimativa na conclusão	EAC	$ETC + ACWP$
Estimativa para concluir	ETC	$\frac{BAC - BCWP}{CPI}$

Organizações que tradicionalmente praticaram este estilo de EVM de gestão de projeto, têm ligado o EVM com a abordagem *Waterfall*. Embora o EVM possa, a princípio, parecer complicado ou difícil de entender, é essencialmente uma técnica formal para a compreensão das seguintes perguntas (Glaiel, 2012):

- O trabalho está a ser realizado conforme o planeado?
- O trabalho está a custar como planeado?
- Qual o custo do trabalho restante?
- Quando será finalizado o projeto?

Segundo Park (2010), o método EVM está focado em completar atividades ao contrário das metodologias ágeis que estão focadas em fornecer capacidades. Park (2010) refere ainda que o método EVM foca nas seguintes estratégias principais:

- Gerir com base nas atividades divididas em pequenas tarefas de duração estimada;
- Gerir riscos ao identificar todas as tarefas necessárias para completar os requisitos do sistema.

Segundo Glaiel (2012), as organizações de engenharia de *software* em larga escala têm tradicionalmente usado abordagens planeadas, pesadas e estilo *waterfall* para o planeamento, execução e monitorização de esforços de desenvolvimento de *software*. Esta abordagem resulta frequentemente em cronogramas de desenvolvimento relativamente longos que são suscetíveis a falhas, especialmente num ambiente de mudança rápida.

Muitos profissionais no mundo do *software* comercial têm lidado com essas pressões adotando a metodologia de desenvolvimento de *software* ágil, uma abordagem projetada para ser flexível e que dá uma boa resposta em ambientes de grandes mudanças (Glaiel, 2012).

As diferentes métricas EVM são:

- EV (*Earned Value*): Custo orçamentado do trabalho realizado (BCWP);
- PV (Planned *Value*): Custo orçamentado do trabalhado programado (BCWS);
- AC (Actual *Cost*): Custo real do trabalho realizado;
- BAC (Budget At Completion) Orçamento na conclusão do projeto;
- SV (*Schedule* Variance): Variação do prazo (EV - PV);
- CV (*Cost* Variance): Variação de custo (EV - AC);
- SPI (*Schedule Performance Index*): Índice de desempenho do prazo: EV / PV (>=1 significa programação do projeto sob controlo);
- CPI (*Cost Performance Index*): índice de desempenho de custo (EV / AC) (>=1 significa custo do projeto sob controlo);
- EAC (Estimate At Completion): Estimativa de conclusão ou custo total esperado do projeto no final (EAC = BAC / CPI);
- ETC (Estimate To Complete): Estimativa para completar, ou custo adicional esperado necessário para concluir o projeto a partir do ponto de cálculo (ETC = EAC - AC).

Segundo Roy & Goutam (2014) o método EVM tem as seguintes vantagens:

- Melhora o processo de planeamento;
- Promove uma definição clara do âmbito do trabalho;

- Estabelece uma clara responsabilidade pelo esforço de trabalho;
- Integra o desempenho técnico, de âmbito e de custo;
- Fornece aviso prévio de problemas potenciais;
- Identifica áreas problemáticas para atenção imediata e facilita uma perspetiva proactiva da gestão;
- Permite relatórios rigorosos dos impactos no custo e cronograma de problemas conhecidos;
- Aumenta a capacidade de avaliar e integrar fatores técnicos, de âmbito, de custo e de risco;
- Fornece uma comunicação consistente e clara do progresso em todos os níveis de gestão;
- Melhora a visibilidade e a responsabilidade do projeto;

Segundo Glaiel (2012) o EVM tem dois problemas:

1. Controlar o planeamento de um projeto de *software* com o EVM tradicional exige um plano inicial completo. Isto significa que o âmbito do projeto deve ser totalmente compreendido, especificado e planeado. Qualquer mudança posterior na direção é considerada como mudança de âmbito e deve passar por um processo formal de mudança contratual. O plano não pode evoluir e mudar à medida que novas informações ou restrições são obtidas. O replaneamento é um caso difícil no EVM. Por outras palavras, um plano rígido não é um plano ágil e, portanto, tira o potencial de "agilidade" do projeto.

2. As etapas do EVM são arbitrariamente discutidas. Por exemplo, o EVM reivindica uma parcela significativa do "valor obtido" num projeto de desenvolvimento após os requisitos e as fases do projeto serem concluídas. Um projeto pode, portanto, reivindicar estar com uma conclusão de 50%, embora nenhum código ainda tenha sido produzido. Por outro lado, os projetos *Agile* acompanham o progresso por *feature,* a percentagem do projeto concluído é indicada com mais precisão com base no número *features* terminadas. Esta é também uma proposta mais valiosa para o cliente, uma vez que 50% completa significa que eles podem potencialmente receber uma entrega de *software* com 50% da funcionalidade já existente.

3.5 Metodologias ágeis

O movimento ágil (formalmente) começou em 2001, quando 17 líderes da indústria de *software* se reuniram para identificar o terreno comum que une os seus vários pontos de vista. A maioria tinha as suas próprias abordagens para construir *software*, nomeadamente: Programação eXtrema (XP), *Scrum*, Método de Desenvolvimento de Sistemas Dinâmicos (DSDM), Crystal Clear, *Feature* Driven Development (FDD) e outras abordagens. A intenção não era padronizar seus esforços numa única abordagem, mas sim documentar alguns valores e princípios comuns com os quais todos concordavam. Esses valores foram documentados no Manifesto Ágil, que se tornou o "documento fundador" do movimento ágil de *software* (Rusk, 2009).

Segundo Fowler & Highsmith (2001), os doze princípios do manifesto ágil são:

- A nossa maior prioridade é satisfazer o cliente através da entrega rápida e contínua de *software* com valor para o negócio;
- Receber de braços abertos a mudança de requisitos, mesmo tarde no desenvolvimento. Os processos ágeis aproveitam a mudança para a vantagem competitiva do cliente;
- Entregar o *software* funcional frequentemente, de um par de semanas a um par de meses, com uma preferência à escala de tempo mais curta;
- Clientes e equipas de desenvolvimento devem trabalhar juntos diariamente durante todo o projeto;
- Criar projetos em torno de indivíduos motivados. Dê-lhes o ambiente e o apoio que necessitam e confie neles para a realização do trabalho;
- O método mais eficiente e eficaz de transmitir informações para uma equipa de desenvolvimento e dentro da mesma, é a conversação cara-a-cara;
- O *software* funcional é a principal medida do progresso;
- Os processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, equipa de desenvolvimento e utilizadores devem manter um ritmo constante;
- A atenção contínua à excelência técnica e ao bom *design* aumenta a agilidade;
- Simplicidade - a arte de maximizar a quantidade de trabalho não essencial não feito - é essencial;
- As melhores arquiteturas, requisitos e projetos emergem de equipas auto-organizadas;
- Em intervalos regulares, a equipa reflete sobre como se tornar mais eficaz, em seguida, ajusta o seu comportamento em conformidade.

As organizações que produzem produtos de software procuram adaptar os seus processos, utilizando metodologias de desenvolvimento ágil para promover a qualidade dos seus produtos (Oliveira et al., 2014).

O desenvolvimento de *software* ágil usa um modelo empírico para simplificar a complexidade do trabalho de gestão. A medição contínua é inerente a esta abordagem. Os primeiros projetos ágeis foram liderados por "inovadores", as pessoas à vontade com a mudança e risco, que começaram pequeno e navegaram intuitivamente neste território desconhecido (Hartmann & Dymond, 2006).

Os projetos de *software* ágil partem dos seguintes pressupostos (Cabri & Griffiths, 2006):

- O trabalho do projeto é concluído iterativamente e não é sequencial e linear. Por exemplo, o feedback de cada iteração afeta o próximo; o trabalho complexo pode ser realizado anteriormente no ciclo de vida para reduzir o risco da abordagem arquitetónica, etc.
- O âmbito é definido com um nível alto no início de um projeto. Apenas a próxima iteração é especificada com mais detalhes.

- As mudanças de âmbito são esperadas e frequentes ao longo do projeto, como normalmente os utilizadores de *software* só entendem o que querem quando veem algo tangível e o Ágil procura responder a isso.

Técnicas de medição herdadas de metodologias anteriores podem dificultar equipas ao tentar fazer a mudança para o alto desempenho com métodos ágeis (Hartmann & Dymond, 2006).

A promessa do ágil é que ajudará as organizações de desenvolvimento de *software* a cortar custos e encurtar os tempos de desenvolvimento. Processos pesados de engenharia de *software* tipo *waterfall,* são considerados culpados por alguns especialistas da área, por inflacionar os custos e a duração dos projetos de *software*. Pode-se argumentar que essas práticas de desenvolvimento carregadas de processos redirecionaram o foco das equipas de *software* para o controlo de processo estatístico, impulsionado pelo desejo de alcançar a previsibilidade do projeto, o Modelo de Maturidade de Capacidade (CMMi) encaixa-se nos modelos estabelecidos de gestão de projetos (Glaiel, 2012).

Nos últimos anos, as metodologias Ágeis apareceram como uma reação às formas tradicionais de desenvolvimento de *software* e reconhecem a necessidade de uma alternativa aos processos de desenvolvimento de *software* pesados e documentados (Ilieva et al., 2004).

A gestão ágil de custos fornece etapas simples ou processos de comparação com a gestão de custos tradicional. Uma vez que a maioria das empresas estão a utilizar o método ágil nos seus projetos de desenvolvimento, o custo de gestão ágil torna-se mais importante devido ao aumento da sua utilização que resulta nas vantagens do aumento da eficácia e eficiência dos projetos de desenvolvimento em comparação com a gestão de custos tradicional (Seetharaman & Mansor, 2015).

Para um projeto ser ágil, não só deve empregar métodos de desenvolvimento ágil, mas também deve adotar medidas coerentes com um desenvolvimento de um produto ágil. A organização de desenvolvimento deve estar disposta a praticar *refactoring*, caso contrário irá perder benefícios. O *software* em si deve ser ágil, comprometendo-se a entregas incrementais rápidas e, portanto, deve ser arquitetado de acordo com os conjuntos de recursos disponíveis (Glaiel, 2012).

A metodologia ágil reconhece que tanto os requisitos como o âmbito do projeto vão mudar. É fácil aceitar as mudanças se estas poderem ser incorporadas em ciclos curtos (semana / mês). Num projeto ágil, as solicitações de mudança podem ser incorporadas nas iterações futuras em questão de semanas ou meses. Os métodos ágeis podem acomodar essas mudanças devido ao seu ciclo curto de entregas (Roy & Goutam, 2014).

As equipas de desenvolvimento de *software* que utilizam o "desenvolvimento ágil" na prática utilizam uma variedade de "métodos ágeis". Essas práticas são anunciadas para reduzir custos de coordenação, para

concentrar equipas e produzir iterações de produto estáveis que podem ser entregues de forma incremental. Desenvolvimento ágil de *software* tornou-se uma abordagem reconhecida para a engenharia de sistemas de *software* no mundo comercial (Glaiel, 2012).

Métodos ágeis mostram muitos benefícios em diferentes áreas de desenvolvimento de *software* como produtividade, visibilidade do projeto, qualidade do *software* e muitas áreas mais. Além disso, no método ágil, os requisitos e detalhes do projeto são flexíveis e podem ser alterados em qualquer fase da fase de desenvolvimento (Seetharaman & Mansor, 2015).

Os métodos ágeis enfatizam o planeamento estruturado, multinível de planeamento, o envolvimento contínuo do cliente, o teste frequente (se não contínuo), a integração contínua e a entrega frequente de *software* potencialmente dedutível (Palmquist et al., 2013).

O movimento ágil agora está espalhado por todo o mundo. Tem sido introduzido em todos os tipos de Indústrias com grande popularidade entre as várias equipas de desenvolvimento. Adotar métodos ágeis significa incorporar novas abordagens dinâmicas em termos de papéis, estilo de gestão, monitorização e controlo de projetos de desenvolvimento de *software* (Kane, 2007).

Segundo Kantor et al. (2016), as metodologias ágeis são as melhores práticas atuais no desenvolvimento de *software*. As metodologias ágeis são favorecidas, entre outras razões, permitindo novos planos e novas priorizações frequentes do próximo trabalho de desenvolvimento baseado em resultados recentes e *backlog* atual (Kantor et al., 2016).

As empresas estão a mover-se para os métodos ágeis porque o mercado exige mais flexibilidade, uma reação mais rápida e maiores expectativas de alta qualidade. Métodos ágeis prometem uma entrega mais rápida e menos despesas no desenvolvimento de *software* (Kane, 2007).

Os testes de desempenho são uma atividade essencial no ciclo de vida do desenvolvimento de *software*. O desempenho da aplicação é a principal consideração em todas as iterações do processo de desenvolvimento ágil e impulsiona o desenvolvimento de melhores versões de *software* (Vani et al., 2014).

A seleção de políticas de gestão e a combinação de práticas ágeis por organizações de desenvolvimento de *software* necessitam de ser equilibradas para uma otimização do sistema (Glaiel, 2012).

Segundo Palmquist et al. (2013) os métodos ágeis têm os seguintes princípios:

- envolvimento contínuo do utilizador;
- múltiplos incrementos ou *release*s de capacidade rapidamente executados;
- prototipagem precoce e sucessiva para apoiar uma abordagem evolutiva;
- abordagem modular de sistemas abertos.

Segundo Park (2010), *Agile* está focado em fornecer capacidades para:

- gerir com base na entrega de recursos programados em caixas de tempo fixas;
- identificar as tarefas necessárias para entregar cada *user story*;
- gerir riscos ao identificar todas as tarefas necessárias para completar as capacidades.

Palmquist et al. (2013) ilustra a forma como uma abordagem ágil poderá ser implementada, mas refere que é apenas uma abordagem recomendada não um guião para a sua utilização.

Globalmente:

- Depois que o cliente selecionar um contratado de desenvolvimento, o cliente e o contratado concordam com o âmbito geral do projeto, os objetivos do sistema, o orçamento e o âmbito;
- O cliente e o contratante de desenvolvimento criam, em conjunto, um plano ou *roadmap* de alto nível;
- Ambas as partes concordam que o processo de desenvolvimento de *software* do contratante de desenvolvimento - por mais eficiente que seja - pode ser gerido, mas não totalmente planeado (ou seja, os riscos ocorrem);
- Os requisitos de alto nível são desenvolvidos colocando cada um num contexto de ambiente, inputs, produtos, etc. para facilitar a compreensão e comunicação;
- O contratado de desenvolvimento refina as estimativas de quanto tempo e quanto levará para construir a capacidade ou o conjunto de recursos necessários para implementar um requisito / *user story*; Além disso, ele assume os riscos associados a cada um;
- O *Product Backlog* é preparado para equilibrar as prioridades (o que os utilizadores precisam primeiro) e os riscos do contratante de desenvolvimento (quais riscos e problemas devem ser explorados e resolvidos para que a construção, implantação e operação do sistema sejam bem-sucedidas);
- Usando o *backlog* de produtos priorizados, o produto final é então dividido numa série de *release*s;
- Os requisitos são alocados à luz de uma visão transversal dos requisitos arquitetónicos de alta prioridade abrangentes que devem ocorrer para alcançar o sucesso das iterações;
- Os requisitos / histórias de maior prioridade são alocados para a primeira *release* para criar o seu *backlog*, com os seguintes requisitos de prioridade mais altos alocados para a segunda versão para ser criado o *release backlog*, e assim por diante;
- Cada *release* tem um plano de lançamento, com versões anteriores com detalhes maiores do que versões posteriores;
- As *release*s são divididas numa ou mais iterações (*sprints*) que podem variar entre duas semanas a dois meses, mas duas a quatro semanas é o mais típico;

- As iterações são deliberadamente mantidas em blocos de tempo curtos e geridas para melhorar o planeamento e manter o foco da equipa, bem como para permitir demonstrações frequentes de produtos e melhoria de processos em equipa;
- As estimativas de custo e âmbito são usadas para garantir que o trabalho alocado à iteração seja realista;
- As iterações são divididas numa série de tarefas de trabalho diário (às vezes chamadas de tarefas) para a equipa de desenvolvimento.

Durante o trabalho de cada dia:

- A equipa realiza uma reunião de stand-up diária, onde cada membro da equipa apresenta o trabalho que completou, o que vai realizar em seguida e identifica qualquer problema ou obstáculo que enfrenta;
- O código base é continuamente integrado e muitas vezes testado, pelo menos uma vez por dia e este código base deve passar todos os testes, após a integração.

Durante cada iteração:

- O cliente não pode adicionar mais requisitos à iteração, mas pode esclarecer os requisitos já atribuídos à iteração;
- A equipa mantém boas apresentações visuais de progresso e problemas para manter todos os membros da equipa informados;
- O *release backlog* é preparado e o plano para a próxima iteração é refinado para permitir o trabalho ininterrupto.

No final de cada iteração:

- O contratante de desenvolvimento demonstrará ou oferecerá alguma capacidade útil para o cliente;
- Uma vez que o cliente está satisfeito, a capacidade está preparada para ser lançada ou transferida para a próxima iteração (dependendo do plano de lançamento);
- Se for o caso, os materiais de treino e a documentação estão completos;
- Se houver alguma capacidade inacabada, eles são colocados de volta no *backlog*. A nova priorização pode ocorrer porque as prioridades podem ter mudado e, dependendo da prioridade "nova", elas poderiam ser incluídas na próxima iteração;
- O cliente possui uma retrospetiva para rever o que funcionou e o que não funcionou;
- O contratante de desenvolvimento fornecerá pelo menos uma capacidade útil a cada *release*.

As versões e iterações continuam até:

- Todas as capacidades desejadas serem entregues.

3.5.1 Scrum

Scrum é uma abordagem ágil bem conhecida para o desenvolvimento rápido de soluções de *software*, mas contém complexidades de integração. As complexidades de integração diminuem a qualidade do produto global. Portanto, há uma necessidade de melhorar a garantia da qualidade do processo do modelo *Scrum* para desenvolver um produto mais confiável e flexível com maior satisfação do cliente (Munawar & M., 2015).

Scrum é uma estrutura de gestão de projetos ágil. Este *framework* concentra-se especificamente na maximização do retorno sobre o investimento (ROI). *Scrum*, no entanto, não define como gerir e acompanhar os custos para avaliar o ROI real contra a visão. Para avaliar o ROI real, os custos do projeto devem ser avaliados e usados para determinar se os planos de lançamento têm valor comercial suficiente ou se é necessário um planeamento adicional (Sulaiman et al., 2006).

Usando o método *Scrum*, as mudanças no processo do ambiente de desenvolvimento do sistema e variáveis técnicas como requisitos, prazo, recursos e tecnologia são flexíveis. Isso tornará o processo complexo e imprevisível e permitirá uma resposta às mudanças e, assim, tornará o processo de desenvolvimento do sistema melhor o que deverá resultar num produto útil para os utilizadores (Seetharaman & Mansor, 2015).

Um desafio ao avaliar o ROI é que o *framework Scrum* não integra os custos de integração e deixa isso para os implementadores do *Scrum* (Sulaiman et al., 2006).

O desenvolvimento ágil recebe mais apreciação do mercado devido à sua natureza flexível e alta produtividade. O *scrum* fornece melhor gestão dos processos em comparação com outros modelos ágeis. A *framework Scrum* tem várias características e pontos fortes, mas ainda peca em práticas de engenharia e qualidade (Munawar & M., 2015).

Segundo Downey & Sutherland (2013), as dez métricas essenciais para a medição de um desenvolvimento de *software* utilizando a frame*work agile Scrum* são:

- Velocidade;
- Capacidade de trabalho;
- Fator de Foco;
- Percentagem de trabalho adotado;
- Percentagem de trabalho encontrado:
- Precisão da estimativa;
- Precisão da previsão;
- Aumento do valor alvo;
- Sucesso em escala;

- Registo de ganhos e perdas.

Vários autores têm analisado as vantagens e desvantagens da metodologia *Scrum*. Segundo Mahalakshmi & Sundararajan (2013) as vantagens do *Scrum* são:

- Permite satisfazer o cliente, otimizando o tempo de resposta e a capacidade de atender às solicitações;
- Aumenta a qualidade;
- Aceita e antevê as alterações;
- Fornece melhores estimativas enquanto gasta menos tempo criando-as;
- Tem mais controlo sobre o calendário do projeto
- É ideal para mudanças rápidas, acumulando requisitos;
- Traz benefícios para o cliente e gestor de projeto;
- É rápido e pode adaptar as mudanças facilmente;
- Congela o cronograma/calendário - *Sprint* curto seguido de *Sprint* curto;
- Boa estimativa do âmbito;
- Nunca altera o cronograma ou *Sprint* ajusta o âmbito, se necessário, para atender às datas de lançamento;
- Estimativas de trabalho muito mais fáceis;
- Trabalho prossegue e termina mais logicamente.

3.5.2 eXtreme Programming

Segundo Beck (1999), a eXtreme Programming, também conhecida como XP, é uma metodologia para equipas de pequeno a médio porte que desenvolvem *software* face a exigências vagas ou em rápida mudança.

As práticas de desenvolvimento das equipes XP definidas por Beck (1999) são regidas pelas seguintes regras:

- O jogo de planeamento. Determinar rapidamente o âmbito da próxima versão, combinando prioridades de negócios e estimativas técnicas. À medida que a realidade ultrapassa o plano, atualiza-se o plano;
- Pequenas entregas. Colocar um sistema simples em produção rapidamente, em seguida, lançar novas versões com um ciclo muito curto;
- Metáfora. Guiar todo o desenvolvimento com uma simples história compartilhada de como todo o sistema funciona;

- *Design* simples. O sistema deve ser concebido da forma mais simples possível num dado momento. Complexidade extra é removida assim que é descoberta;
- Testar. Os programadores continuamente escrevem unidades de teste, que devem ser executadas sem falhas para o desenvolvimento continuar. Os clientes escrevem testes demonstrando que os recursos estão concluídos;
- *Refactoring*. Os programadores reestruturam o sistema sem alterar seu comportamento para remover a duplicação, melhorar a comunicação, simplificar ou adicionar flexibilidade;
- Programação por pares. Todo o código de produção é escrito com dois programadores numa máquina;
- Propriedade coletiva. Qualquer pessoa pode alterar qualquer código em qualquer lugar do sistema a qualquer momento;
- Integração contínua. Integrar e construir o sistema muitas vezes ao dia, idealmente, sempre que uma tarefa é concluída;
- 40 horas por semana. Não ultrapassar as 40 horas por semana como regra. Nunca trabalhar horas extras em duas semanas consecutivas;
- Cliente no local. Inclua um utilizador real e ao vivo na equipa, disponível em tempo integral para responder a perguntas;
- Padrões de codificação. Os programadores escrevem todo o código de acordo com as regras com enfâse na comunicação através do código.

3.5.3 Test Driven Development

Segundo Beck (2001), TDD foi inspirado pela filosofia "teste primeiro" do XP, o Test Driven Development (TDD) inicia o processo de desenvolvimento codificando casos de teste automatizados para as funcionalidades de *software* que serão produzidas. Em seguida, o código de produção é desenvolvido iterativamente até que todos os testes sejam aprovados. Após cada ciclo (iteração), todos os testes são reexecutados para garantir que a nova funcionalidade é bem integrada.

Benefícios de uma abordagem TDD segundo Beck (2001):

- Encoraja a ser explícito sobre o âmbito da implementação;
- Ajuda a separar entre o *design* lógico, o *design* físico e a implementação;
- Cresce a confiança no funcionamento correto do sistema à medida que o sistema cresce;
- Simplifica os projetos.

3.5.4 Feature Driven Development (FDD)

Segundo Highsmith (2002), o desenvolvimento orientado para funcionalidades (FDD) é descrito como tendo apenas o processo suficiente para garantir escalabilidade e repetibilidade e estimular a criatividade e inovação ao longo do caminho

As práticas FDD segundo Highsmith (2002), são:

- Modelação de objetos de domínio: desde que o FDD foi desenvolvido originalmente em 1997 para um projeto baseado em linguagem Java, é adaptado para uma abordagem orientada a objetos. O FDD preconiza a construção de diagramas de classe para capturar os atributos e relações entre os objetos significativos no espaço do problema;
- Desenvolvimento por *feature:* o sistema é dividido num conjunto de *features* que podem ser desenvolvidos de forma incremental;
- Propriedade individual da classe: O FDD pede que cada classe seja atribuída a um indivíduo que seja o responsável final por ela;
- Equipas de *features:* as funcionalidades são desenvolvidas por equipas compostas de proprietários das *features* e uma combinação dos proprietários de classes necessárias para implementar a *feature* pedida;
- Inspeções: revisões formais de código são realizadas para prevenir defeitos e garantir a qualidade;
- Gestão de configuração: o uso do controlo de origem e controlo da versão;
- Relatórios e visibilidade dos resultados: o progresso de cada recurso baseia-se na conclusão dos *milestones* de desenvolvimento. O progresso dos conjuntos de features é relatado regularmente.

3.5.5 Crystal

Segundo Highsmith J. (2002), a combinação do tamanho da equipa e da criticidade direciona um determinado esforço de desenvolvimento para uma metodologia Crystal correspondente. Existem apenas duas regras que regem a prática da família de métodos Crystal.

1. Os ciclos incrementais não podem exceder quatro meses.
2. Os *work*shops de reflexão devem ser realizados após cada entrega para que a metodologia se auto adapte.

Crystal concentra-se sobre pessoas, interação, comunidade, habilidades, talentos e comunicação sendo estas as principais características que influenciam o desempenho. O processo continua importante, mas passa para segundo plano (Highsmith J., 2002).

3.6 *Waterfall* e *Agile*

O que às vezes é perdido nas discussões do Mundo Tradicional / Mundo *Agile* é o fato de que ambos os grupos têm o mesmo objetivo - entregar um produto de qualidade eficiente, de forma responsável e num tempo previsto. Ambos os mundos fazem os mesmos "tipos" de coisas: definem, reúnem, analisam, projetam, codificam, testam, entregam, mantêm e libertam. A forma como eles fazem essas coisas é que é diferente (Palmquist et al., 2013).

Tanto o mundo tradicional como o mundo ágil também trabalham com os mesmos blocos de programação programáticos básicos (Palmquist et al., 2013):

- Âmbito;
- Custo;
- Cronograma;
- Desempenho.

Além disso, tanto o mundo tradicional como o mundo ágil usam os mesmos blocos de construção técnicos ou de desenvolvimento (Palmquist et al., 2013):

- Análise do requisito;
- Capacidade de *design* para satisfazer o requisito;
- Construir a capacidade;
- Testar a capacidade para garantir que o requisito seja cumprido;
- Implementar a capacidade.

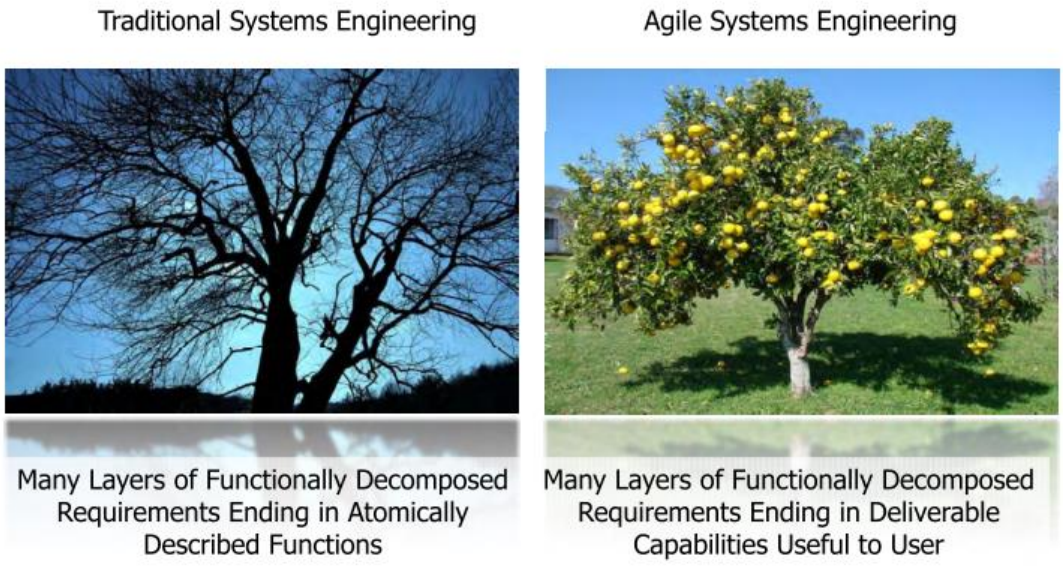

Figura 5 - Métodos tradicionais vs Métodos ágeis (Park, 2010)

Segundo Park (2010) o método tradicional (Figura 5Figura 5 - Métodos tradicionais vs Métodos ágeis *(Park, 2010)*).

- Gere com base nas atividades divididas em tarefas de pequena duração;
- Gere os riscos ao identificar todas as tarefas necessárias para completar os requisitos do sistema.

Ágil está focado na entrega de capacidades

- Gere com base na entrega de *features* agendadas em caixas temporais fixas;

- Identifica tarefas necessárias para entregar cada história;
- Gere riscos ao identificar todas as tarefas necessárias para completar as capacidades.

Segundo Cabri & Griffiths (2006) os projeto tradicionais seguem os seguintes pressupostos:

- O âmbito é bem compreendido e pode ser totalmente definido no início de um projeto. Portanto, enquanto um processo de controlo de mudanças está normalmente pronto, esperam-se algumas mudanças de âmbito;
- O trabalho do projeto é completado de forma sequencial e naturalmente linear, onde as taxas de progresso atuais são indicadores das taxas futuras.

Enquanto metodologias de desenvolvimento tradicional se concentram na geração de documentos do projeto e na realização rigorosa dos processos, a proposta *agile* foca-se no próprio desenvolvimento do projeto e no relacionamento dos participantes (Vasconcelos, Carvalho, Henrique, & Mello, 2011).

Existe, definitivamente, uma falta de alinhamento entre as metodologias / ferramentas usadas pela gestão de projetos tradicional e as introduzidas pelas técnicas *Agile*. O papel do Gestor de Projetos continua a ser muito crítico na adoção e aplicação da agilidade em projetos. Ele ainda precisa de manter o controlo sobre tarefas diferentes e concentrar-se no valor a ser pago pelo que a equipa concretizou (Kane, 2007).

Projetos tradicionais determinam o âmbito logo no início do projeto, a mudança de âmbito é pouco frequente. Nos projetos de *software Agile*, o âmbito inicial não é assumido como completo. O âmbito é elaborado à medida que as iterações do projeto são concluídas, com base no feedback do utilizador e dos *stakeholders* no final de cada iteração (Cabri & Griffiths, 2006).

Os projetos tradicionais seguem uma progressão linear ao longo do ciclo de vida do projeto (Tabela 5Tabela 5 - Waterfall vs Scrum *(Despa, 2014)*). Projetos similares já foram feitos antes, estão bem definidos e, portanto, têm menos risco. Os projetos de *software* ágil, por outro lado, são tipicamente sem precedentes, podem exigir investigação & desenvolvimento, um maior nível de criatividade e, portanto, o trabalho geralmente prossegue de forma não linear. À medida que o verdadeiro âmbito do projeto fica claro, pode ser necessário algum retrabalho dos elementos inicialmente concluídos (Cabri & Griffiths, 2006). O alcance de um projeto de *software* ágil não pode ser totalmente definido de forma ascendente no início, como é feito em projetos tradicionais (Cabri & Griffiths, 2006).

Tabela 5 - Waterfall vs Scrum (Despa, 2014)

Metodologia	Características	Pontos Fortes	Pontos Fracos
	- Documentação abrangente	- Fácil de gerir	- O código do sistema é entregue no final do projeto

Waterfall	- Planeamento meticuloso - Processo linear e sequencial - Cada fase tem os seus próprios deliverables	- Fácil de entender para o proprietário e para a equipa de projeto	- Não lida facilmente com mudanças de requisitos - Baixa tolerância ao *design* e erros de planeamento
Scrum	- Desenvolvimento iterativo - Abordagem timebox - Reuniões diárias para avaliar o progresso - Auto-organização da equipa de desenvolvimento - As tarefas são geridas usando *backlog*s;	- Entregar produtos em ciclos curtos - Permite um feedback rápido - Adaptação rápida à mudança	- Falta de documentação - Requer experiência de desenvolvedores - Dificuldade em estimar no começo o esforço geral necessário para implementar grandes projetos - As estimativas de custo não são muito precisas

As metodologias tradicionais de desenvolvimento de *software* são dificilmente aplicáveis ao desenvolvimento de sistemas avançados e complexos (Usman, Soomro, & Brohi, 2014).

Atualmente, as melhores equipes de *Scrum* no mundo ganham em média 750% sobre a velocidade de equipas que utilizam a metodologia *waterfall*. Para além da velocidade muito superior, estas equipas *Scrum* desenvolvem um produto com muito maior qualidade, com maior satisfação do cliente e experiência dos profissionais de desenvolvimento (Downey & Sutherland, 2013).

3.7 Situação Atual

Uma das principais diferenças entre os métodos tradicionais e os métodos ágeis é que os projetos tradicionais concentram-se em requisitos, enquanto os projetos ágeis em replaneamento contínuo (Cohen & Gan, 2014).

Segundo Hayes et al. (2014), existem diferenciadores importantes entre as abordagens ágeis e as tradicionais:

Há uma dependência de uma "abordagem time-box" em vez da tradicional abordagem fase-gate para gerir o progresso. Isso significa que a organização de desenvolvimento está a trabalhar para maximizar o trabalho realizado dentro de limites de tempo estritamente definidos. O âmbito permanece fixo e o trabalho a ser realizado é a variável. Isto contrasta com muitas abordagens tradicionais em que o período de desempenho pode estar sujeito a negociação, ao mesmo tempo em que se tenta fixar o âmbito do (s) produto (s) de trabalho a serem entregues. Além disso, o cliente (ou representante do cliente) está envolvido em avaliações frequentes de entregas - ao invés de sucessivos produtos de trabalho intermediário que levam a uma única entrega do *software*.

Um foco na entrega de *software* utilizável substitui a entrega programada de produtos de trabalho provisórios e as cerimónias que normalmente se concentram em *milestones*. Isso significa que o cliente (ou representante do cliente) deve esperar ver o produto a funcionar, frequentemente. As interações e demonstrações planeadas permitem um processo disciplinado para mudar os requisitos e moldar a forma final do produto entregue. O teste de aceitação acontece iterativamente, ao invés de no final.

O sucesso do desenvolvimento de aplicações de *software* tem basicamente 3 pilares (programação, custo e qualidade). O gestor de projeto utiliza o EVM na metodologia *waterfall* para gerir o custo e para os prazos, mas em metodologias ágeis há um preconceito de que o EVM não pode ser implementado (Roy & Goutam, 2014).

Processos ágeis dão a opção de produzir uma definição razoavelmente completa de âmbito no início do projeto. O método EVM utiliza metas globais de âmbito, tempo e custo para que seja possível acompanhar o progresso em direção a essas metas. Para ser ágil essas metas não precisam de ser definidas. Mas para utilizar EVM em modelos ágeis é necessário defini-las (Rusk, 2009).

O conceito de EVM é aplicável a projetos ágeis de *software*. Os projetos ágeis superam as limitações inerentes do EVM que se aplica ao projeto da *waterfall*. Embora a adoção do EVM seja limitada na comunidade ágil, essa aplicação irá tornar-se mais popular com treino, educação e pesquisa adicional (Roy & Goutam, 2014).

Alguns autores já têm vindo a introduzir o método EVM em metodologias ágeis, nomeadamente Hayes et al. (2014), Kane (2007), Rusk (2009), Seetharaman & Mansor (2015), Sulaiman et al. (2006), Torrecilla-Salinas et al. (2015) chamando a essa junção o "*Agile*EVM".

Sulaiman et al. (2006) sugere que o *Agile*EVM se concentra na medição do progresso ao nível da release, e não ao nível de *sprint* ou ao nível do produto, sendo esta a maneira mais apropriada de usar fórmulas de gestão de valor agregado em projetos ágeis. Essas fórmulas podem ser facilmente utilizadas para medir progresso de um projeto com várias entregas, mas isso exigiria que o atraso de várias entregas fosse identificado e estimado (Sulaiman et al., 2006).

Um exemplo de uma aplicação do AgileEVM seria para quando um *Sprint* tem uma entrega para o cliente, utiliza-se o *Agile*EVM para ajudar com todo o ciclo de vida do produto. Em comparação com os requisitos do EVM tradicional, *Agile*EVM alavanca itens de trabalho que são parte integrante do processo *Scrum*. Medimos o progresso no final de cada *sprint*, quando a velocidade real do *sprint* e os custos reais são conhecidos (Sulaiman et al., 2006).

O estudo realizado por Sulaiman et al. (2006) concluiu que a implementação do processo *Agile*EVM não tem impacto negativo na velocidade de uma equipa *Scrum*.

Abordagens EVM simplificadas são relevantes tanto para o mundo ágil como para o mundo clássico do EVM (Rusk, 2009).

Mas nem tudo são vantagens com a aplicação do *Agile*EVM como refere Cabri & Griffiths (2006). A aplicação direta do *Earned Value Management* em projetos ágeis provavelmente resultará num Valor Planeado (PV) inválido no início do projeto, com sobre ou sob execuções ocorrendo durante a execução do projeto.

Em ágil a mudança é esperada e frequente ao longo do ciclo de vida do projeto, assim, medir o progresso em relação ao plano inicial levará a erros. Embora existam problemas com a tentativa de aplicar *Earned Value* a projetos ágeis, técnicas de gestão de projeto ágeis, como gráficos de burn-down (indicando a quantidade de funcionalidades pendente vs. concluída ao longo do tempo, etc.) fornecem informações de status e progresso muito semelhantes às tentativas de medição do *Earned Value* (Cabri & Griffiths, 2006).

Os custos podem ser adicionados aos gráficos para exibir as informações juntamente com a taxa de conclusão de *features*. Tudo isso pode ser mais valioso para o gestor de projeto e para as partes interessadas na monitorização de um projeto ágil, ao invés de tentar aplicar o Eaned *Value* tradicional (Cabri & Griffiths, 2006).

Projetos com ciclos de entrega extremamente curtos não encontrarão tanta utilidade nas técnicas de rastreamento (como o EVM) como projetos com ciclos de entrega mais longos envolvendo múltiplos *sprints* ou iterações. A boa notícia é que as técnicas e ferramentas introduzidas pela gestão de projetos tradicionais podem ser totalmente aplicáveis no mundo ágil quando os requisitos são conhecidos (Kane, 2007).

Segundo Sulaiman et al. (2006), os métodos ágeis não definem como gerir e acompanhar os custos para avaliar a informação esperada sobre o retorno do investimento (ROI). Portanto, a interação dos gráficos de *burn-up* e *burn-down* (como usado no *Scrum*) não fornecem informações de custo de projeto. Métricas ágeis não fornecem estimativas de custo na conclusão do lançamento nem métricas de custo para suportar o negócio quando consideram tomar decisões como alterar requisitos num lançamento. O *AgileEVM* fornece essas informações e, portanto, é uma excelente extensão para as informações fornecidas pelos *Burn Down Charts*.

Apesar de introduzir mais documentação e métricas através da inclusão do método EVM em abordagens ágeis as vantagens superam as desvantagens como a revisão de literatura prova.

4 Fundamentação do *Scrum*

Neste capítulo é apresentada a justificação do porquê da utilização da metodologia ágil *Scrum* nesta dissertação. Foi realizado também uma análise mais detalhada da metodologia assim como a definição de conceitos fundamentais para a utilização do *Scrum*. Foram ainda descritas as áreas de gestão de projetos abrangidas pela metodologia assim como algumas descrições sobre essas mesmas áreas.

Segundo VersionOne (2006), no seu *survey* "State *of Agile*" com quase 1000 inquiridos, as vantagens específicas da adoção de práticas ágeis são: tempo para o mercado acelerado, aumento de produtividade, redução de defeitos de *software* e redução de custos.

Esse *survey* mostra também que nos projetos ágeis, os intervenientes consideram que a taxa de sucesso do projeto é de 81%, com o sucesso caracterizado por duas opções: muito bem-sucedido ou apenas bem-sucedido, um bom contraste com apenas 29% de sucesso dos projetos não ágeis. Mostra ainda que com a adoção das práticas ágeis houve uma melhoria nas competências para a gestão da mudança de prioridades, o aumento da moral da equipa de desenvolvimento, a melhoria na qualidade do *software,* a redução do risco do projeto e o alinhamento entre as TI e os objetivos de negócio. Cada um destes aspetos teve uma melhoria de pelo menos 50%.

O mesmo *survey* demonstrou que as metodologias ágeis tinham a distribuição de uso apresentada na tabela 6Tabela 6 - Utilização das metodologias ágeis:

Tabela 6 - Utilização das metodologias ágeis

Metodologia ágil	Percentagem de Uso
Scrum	40%
XP	23%
Hybrid \ Custom	14%
DSDM	8%

A mesma organização (VersionOne, 2010) realizou um *survey* em 2010 com 4770 pessoas de 91 países diferentes onde 90% dos inquiridos trabalha em empresas que utilizam métodos ágeis, sendo a metodologia de eleição a *Scrum* com 58% de utilização seguido pela metodologia *Scrum* / XP Hybrid com apenas 17%.

Este *survey* conclui que as duas principais razões que levam a falha de projetos com metodologias ágil são a falta de experiência nas metodologias e o choque entre os valores ágeis com a cultura ou filosofia da empresa. Verificou-se que 83% dos inquiridos afirma que os projetos ágeis são tão ou mais rápidos na conclusão do projeto em comparação com as metodologias tradicionais sendo que 66% considera que são mais rápidos os ágeis.

A mesma organização realizou um outro *survey* em 2015 (Manual, 2006), com 3880 inquiridos onde 95% afirmam que as suas organizações aplicam metodologias ágeis. A maior barreira de adoção das metodologias ágeis continua a ser a falta da cultura da organização para a mudança.

Esse *survey* de 2015 confirmou ainda que a metodologia *Scrum* continua a ser a favorita com 58% de utilização e a maior indústria que utiliza as metodologias ágeis continua a ser a de *software*. As principais falhas que levam ao insucesso de projetos que utilizam metodologias ágeis continuam a ser a falta de experiência nas metodologias ágeis e o choque entre os valores ágeis com a cultura ou filosofia da empresa.

Tabela 7 - O efeito de adotar a metodologia Scrum (Wan, Zhu, & Zeng, 2013)

	Aumentou consideravelmente	Não mudou	Desceu
Eficiência da Produtividade	68%	27%	5%
Trabalho em equipa	52%	39%	9%
Adaptabilidade	63%	33%	4%
Responsabilidade	62%	32%	6%
Habilidade de Cooperação	81%	18%	1%

A melhor equipa de *Scrum* no mundo em ganha em média 750% sobre a velocidade de equipas que utilizam o método *waterfall* com muito maior qualidade, satisfação do cliente e experiência de desenvolvedor (Downey & Sutherland, 2013).

4.1 Papéis do *Scrum*

Os papéis propostos pelo *SCRUM* são:

- *Product Owner;*
- *Scrum Master;*
- Team.

Product Owner

O *Product Owner* assume toda a responsabilidade pelo sucesso do projeto e é, em última instância, o responsável máximo perante a equipa, os *Stakeholders* e a empresa. O *Product Owner* é responsável por criar, manter e constantemente priorizar o *Product Backlog*. É também o responsável pela maximização do ROI (*Return On Investment*) e o responsável final sempre que surgirem questões de requisitos do produto. Representa o cliente, e faz a interface com o cliente ao considerar os interesses das partes interessadas. Inspeciona o progresso do produto no final de cada *Sprint* e tem total autoridade para aceitar ou rejeitar o trabalho realizado. Termina um *Sprint* se for determinado que uma mudança drástica na direção é

necessária. O *Product Owner* é a única pessoa que deve enfrentar a responsabilidade se o projeto falhar. Portanto, ele ou ela deve determinar agressivamente quais as características de um produto que são mais importantes, quando elas são desenvolvidas, etc.

Scrum Development Team

Em *Scrum*, uma equipa de desenvolvimento ideal incluiria sete membros, mais ou menos dois. Geralmente, as equipas são compostas por membros multifuncionais, incluindo engenheiros de *software*, arquitetos de *software*, programadores, analistas, especialistas em controlo de qualidade, testadores, *design*ers de *User Interface*, etc.

As principais responsabilidades da equipa *Scrum* são:

- Priorização do *Sprint Backlog*;
- Estimar o esforço para implementar as *User Stories*;
- Desenvolver para atingir metas de *Sprint*;
- Implementar casos de teste;
- Definir testes e unidades de aceitação inicial;
- Identificar obstáculos e informar o *Scrum Master*;
- Auto-organização;
- Reuniões diárias de *Scrum*.

As competências chave que um membro da equipa de desenvolvimento *Scrum* deve ter:

- *Pair Programming;*
- Compreender *TDD*, *BDD*, etc;
- Entender o *Code Smells* e *Re-factoring;*
- Ser focado na integração continua;
- Auto motivado, organizado e com boa capacidade para trabalhar em equipa.

Scrum Master

O *Scrum Master* concentra-se no processo de desenvolvimento e é mentor da equipa *Scrum*. Ele garante que a equipa adere ao processo escolhido e elimina problemas de bloqueio.

As principais responsabilidades do *Scrum Master* são:

- Facilita o processo *Scrum*;
- Dá o compromisso à equipa;
- Tem que acompanhar o progresso e ajudar a equipa a alcançar os compromissos;
- Faz uma análise diária do projeto para que, se necessário impor medidas pró-ativas para serem tomadas com antecedência;

- O objetivo final é fazer com que o *Sprint* seja bem-sucedido;
- É responsável pelos processos de *Scrum* acordados no projeto;
- Cria as regras *Scrum* para o projeto;
- Certifica-se que cada processo é implementado no projeto;
- Cria um ambiente propício à auto-organização da equipa;
- Aplica as time boxes;
- Mantém os artefactos *Scrum* visíveis;
- Serve a equipa e o *Product Owner* para que os requisitos sejam implementados no *software* de trabalho;
- Tem de ser dinâmico e experiente ao desempenhar este papel. Caso o projeto tenha sucesso ou fracasso o *Scrum Master* terá bastante peso nessa responsabilidade;
- Tem a responsabilidade na execução e na entrega do *software*;
- É responsável pela qualidade;
- Tem de resolver impedimentos e conflitos;
- Tem de proteger a equipa de desenvolvimento de interferências externas;
- Fornece feedback à equipa de desenvolvimento;
- Descobre maneiras de melhorar a qualidade, velocidade e valor do negócio.

4.2 Eventos do *Scrum*:

A abordagem *Scrum* propõe os seguintes eventos durante um projeto:

- *Sprint*;
- *Sprint Planning*;
- *Sprint Review*;
- Daily *Scrum Meeting*;
- *Sprint Retrospetive*.

Sprints. Têm uma duração variável (tipicamente entre duas a quatro semanas) e adaptável de projeto para projeto.

Devem ser relativamente curtos para permitir a maior proximidade com o cliente para despoletar a avaliação do trabalho efetuado mais frequentemente de modo a perceber o rumo do projeto ou a corrigir eventuais desvios no produto

Durante a *Sprint* existem vários fundamentos que devem ser respeitados, nomeadamente:

- Não devem ser feitas alterações que alterem a meta da *Sprint*;
- A composição da equipa não deve ser alterada;
- Os objetivos e o nível esperado de qualidade na realização das tarefas não devem diminuir.

Sprint Planning. Todos os ciclos começam com o planeamento. É neste momento que todos os elementos da equipa se juntam e revisitam o *Backlog,* de maneira a ver quais as *User stories* mais urgentes a necessitarem de atenção.

Para que as equipas não se desorientem na discussão sobre o trabalho a efetuar, é aconselhável que a reunião tenha uma duração máxima de 4 horas para uma *Sprint* de duas semanas, sendo que a duração pode aumentar/diminuir proporcionalmente com a duração da *Sprint*. Tipicamente, esta reunião é dividida em duas partes distintas.

Na primeira metade selecionam-se as *User stories* a completar (dentro do *Product Backlog* previamente ordenado por prioridade pelo *Product owner*).

Na segunda parte define-se a maneira como se vai implementar cada *User story*, definindo tarefas específicas e estimando o tempo necessário para cada uma delas.

Daily Scrum Meeting. Para ser eficaz, a sua duração não deve exceder os 15 minutos, deve ser sempre realizada no mesmo local e hora. É um evento que ocorre diariamente e onde cada elemento da equipa responde a três questões:

- O que foi realizado desde a última *Daily Scrum*?
- O que será feito até à próxima *Daily Scrum*?
- Existe algum obstáculo atualmente para a realização da tarefa a decorrer?

Esta reunião é utilizada pelo *Scrum master* para perceber a direção e a velocidade da equipa no *Sprint* que está a decorrer, para os restantes elementos da equipa se aperceberem do estado das tarefas e quais as dificuldades percecionadas pelos elementos que estão a resolvê-las.

Sprint Review. Consiste na reunião que acontece no final de cada *Sprint*, de avaliação e revisão ao trabalho efetuado durante esse ciclo de desenvolvimento. Está presente a equipa de desenvolvimento, o *Scrum master*, o *Product owner* e qualquer outro *stakeholder* que assim o deseje (clientes, gestão de topo, entre outros).

A equipa demonstra o trabalho executado e as *User stories* completas, sendo feita uma análise pelo *Product owner* que, à luz do estabelecido na reunião de planeamento, dá o seu aval e considera (ou não) se as *User stories* respeitam a definição de *Done* acordada no *Sprint Planning*.

Consoante a avaliação e o trabalho efetuado, o *Product owner* pode reordenar ou ajustar o *Product Backlog* para se adaptar a novos contextos.

Sprint Retrospective. A equipa junta-se ao seu *Scrum master* e discute a maneira como a *Sprint* correu em termos operacionais. Aqui se pode discutir a maneira como as reuniões ocorridas decorreram, planear

formas de tornar a aplicação do *Scrum* mais eficiente ou analisar outras causas externas à equipa, mas que podem estar a condicionar o desempenho da mesma. Desta reunião podem surgir um conjunto de melhorias que podem ser integradas no próximo *Sprint* de maneira a tornar a equipa mais eficiente ou melhorar a qualidade do produto desenvolvido.

4.3 Artefactos do *Scrum*:

O *Scrum* propõe o desenvolvimento dos seguintes artefactos:

- *Product Backlog;*
- *Sprint Backlog;*
- *Sprint burndown chart;*
- *Release burn up chart;*

Product Backlog: É o local onde estão ordenados e documentados todos os requisitos funcionais do produto, tanto os implementados como os por implementar. Artefacto de responsabilidade do *Product owner* e gerido com a ajuda do *Scrum master*, deve estar sempre visível e disponível para consulta da equipa ou de outros *stakeholders* interessados.

Os requisitos estão apresentados, tipicamente, em formato de *User stories*. Uma *User story* é uma frase em que se especifica o que se deseja ser implementado e qual o actor que se relaciona com esse requisito. Para cada *User story* deve ser definido a sua medida de complexidade de implementação (normalmente definida como *Story points*) e a sua definição de *Done* (características que permitem ao *Product owner* considerar uma determinada *User story* como implementada ou não).

Sprint Backlog: É o conjunto de itens do *Product Backlog* - *User stories* (e respectivas tarefas) - que a equipa se comprometeu a implementar para uma determinada *Sprint*.

A sua criação é da responsabilidade exclusiva da equipa durante a segunda fase da reunião de planeamento, embora a sua manutenção possa ser feita pelo *Scrum master*.

Sprint burndown chart: Artefacto que está associado ao *Sprint Backlog*, é a visualização em gráfico da quantidade de trabalho (*Story points*) que falta executar até ao final do *Sprint*. Relaciona os *Story points* completos com os planeados.

Release burn up chart. Artefacto que está intrinsecamente associado ao *Sprint burndown chart*, é a visualização em gráfico da quantidade de trabalho (*Story points*) que falta executar até ao final da *Release*. Faz a acumulação dos *Sprint burndown charts*, das *Story points* realizadas em cada *sprint* e o total de *Story points* planeados até à próxima *Release*.

Serão apresentados alguns conceitos essenciais e diretamente ligados ao *Scrum* tais como *features* e *user stories*.

User story pode ser definida como uma pequena funcionalidade que fornece valor ao cliente ou a um utilizador do sistema. Uma *User story* representa certas necessidades dos utilizadores, mas não uma documentação exaustiva delas. Ele atua como um lembrete e os seus detalhes são descobertos durante o processo de colaboração que ocorre para desenvolvê-lo num determinado *Sprint*. Os detalhes das *User story*s geralmente são gravados num conjunto de testes que é usado para verificar se a *Story* está concluída. Uma *User story* boa é caracterizada por um conjunto de atributos, conhecido como o acrónimo *INVEST*: *Independent, Negotiable, Valuable, Estimable, Small* e *Testable* (Torrecilla-Salinas et al., 2015).

(I)ndependent – Independente. As *User stories* devem ser independentes umas das outras. Isto significa que elas devem estar escritas de uma maneira que qualquer uma pode ser implementada a qualquer altura, não estando dependentes da implementação anterior ou posterior de qualquer outra *User story*;

(N)egotiable – Negociável. Baseando-se no pressuposto de que o *Scrum* potencia o trabalho de equipa e a colaboração/comunicação entre todas as partes envolvidas, também as *User stories* podem ser sujeitas a alterações desde a altura em que são criadas até à sua implementação.

(V)aluable – Valiosa. Todas as *User stories* implementadas devem significar um acréscimo tangível para o produto. Para se conseguir esse objetivo, é necessário a colaboração entre o cliente e a equipa de desenvolvimento para perceber se o que se vai implementar vai acrescentar valor ao produto, ou pelo contrário, pode ser um fator de geração de problemas para o produto ou para o utilizador final. Apesar do foco no cliente ser importante, por vezes é necessário pensar em fatores não funcionais que, não sendo percetíveis diretamente ao utilizador, beneficiam-no ao induzir melhorias no funcionamento do produto.

(E)stimable. Estimável. Um dos pontos importantes no *Scrum* é que os *stakeholders* interessados num determinado projeto podem ter a capacidade de saber, em tempo real, qual o esforço necessário para a implementação de determinada *User story*. Essa perceção pode ajudar na priorização do *Backlog* e na decisão sobre o que se vai implementar de seguida.

(S)mall. Curta. Outro fator fundamental para eficiência do processo é projetar *User stories* que, mantendo o valor para o produto, sejam passíveis de implementar num espaço de tempo relativamente curto.

(T)estable. Testável. Um detalhe fundamental para cada *User story* é que ela deve ser escrita e implementada de maneira a poder ser testada e que, assim, seja assegurado ao cliente que a implementação foi bem conseguida.

O conjunto de *User stories* que sigam os princípios INVEST são, por si só, uma segurança para que a sua implementação tenha todas as condições para ser executada corretamente. É importante que todos os

stakeholders tenham bem presente estes princípios na altura da escrita das mesmas, de modo a facilitarem o trabalho às equipas de implementação.
Ao implementar o *Scrum*, é, portanto, essencial compreender o *Scrum* como um ecossistema de partes interdependentes. A coordenação das peças requer inspeção diária para manter um alto estado de energia (Downey & Sutherland, 2013).

Definição de *feature* (Kane, 2007).

No desenvolvimento ágil, uma *feature* é um pedaço de funcionalidade que oferece valor comercial. *Features* podem incluir adições ou mudanças na funcionalidade existente. É a unidade principal para planear e estimar itens de trabalho e deve ter os seguintes critérios:

- Deve fornecer valor comercial;
- Deve ser estimável - deve ter uma definição suficiente para que a equipa de desenvolvimento forneça uma estimativa do trabalho envolvido na sua implementação;
- Deve ser pequeno o suficiente para se encaixar dentro de uma iteração;
- Deve ser testável - entender qual teste automático ou manual que uma *feature* deve passar para ser aceitável para o cliente.

Definição da unidade de estimativa (Kane, 2007):

- Tempo ideal: quanto tempo uma tarefa ocorre sem haver interrupções;
- *Story point*: medição relativa entre *User story*s;
    - Requisitos do utilizador;
    - Lista de *features*;
    - Cenários de casos de uso.
- Velocidade: quantas unidades de estimativa são completadas por uma equipa numa única iteração.

Usman et al. (2014) descreveu as práticas e processos abrangentes pela metodologia *Scrum:*

Gestão do Âmbito

- Desenvolve uma lista de itens de *backlog* do produto e prioriza-os;
- Escolha os itens de *backlog* do produto para a próxima iteração;
- Desenvolve uma estrutura de desdobramento do trabalho para a próximo *release*;
- Além disso, divide-o em pequenas tarefas individuais por cada *sprint*;
- Gere através do *backlog* do produto e o *Product Owner* protege o *sprint*.

Gestão de Tempo

- Os cenários são selecionados para o próximo *sprint* pela equipa; as tarefas são definidas para a realização das *features*;
- A equipa *Scrum* é conduzida durante reuniões de planeamento de *sprint*;
- A estimativa de tarefa é realizada para cada User *story*;
- Desenvolvimento do cronograma geral para a *release*;
- As *features* para os *sprints* são estimadas;
- A estimativa é feita com base em dados empíricos também chamados de velocidade da equipa;
- A equipa é responsável por gerir as particularidades de cada de *sprint*.

Gestão de Custos

- A estimativa de custo *top-down* é realizada para *releases* e *sprints*, usa conceitos como a velocidade do projeto, dias ideais, analogia, opinião de especialista ou desagregação;
- A estimativa de custo de base do *sprint* é calculada;
- O cálculo adicional nas estimativas depende de mudanças, funcionalidades esotéricas / extras e tecnologia moderna;
- O *buffer* de recurso ou agendamento é adicionado;
- A linha de base de custo é criada;
- A linha de base de custo é revista após dois *sprints* (quando a velocidade real da equipe é conhecida);
- *O Product Burndown Chart* é usado como ferramenta de controlo de custos.

Gestão da Qualidade

- Considera os princípios ou valores de *Scrum* para desenvolver e construir a equipa;
- Fomenta a auto-organização na co-localização da construção de equipas;
- Facilita e orienta a equipa na auto-organização, fornecendo feedback contínuo;
- A qualidade confiável é mantida nas práticas de *Scrum* (definição de *Done*, *software* a trabalhar, teste precoce e frequente, remoção de impedimentos, padrões de codificação/teste e métricas);
- A equipa *Scrum* é responsável pela alta qualidade no projeto;
- A garantia da qualidade em *Scrum* é realizada pela equipa de desenvolvimento, auditorias externas podem ser executadas como um *sprint* extra (*sprint* N + 1) para regular os requisitos de conformidade.

Gestão de Recursos Humanos

- O tamanho da equipa depende da natureza e agenda do projeto, bem como do orçamento;

- Planeamento para sete (mais ou menos 2) membros totalmente dedicados;
- Distribui o projeto em pequenas equipas se o âmbito for grande;
- Inicialmente, cria uma equipa interfuncional que é mantida até ao final do projeto.

Gestão de Comunicação

- Define os *stakeholders*, nomeando um *Product Owner* para a equipa *Scrum*;
- O estado do projeto pode ser medido por *release*s / *sprint backlog*s ou por *Burn up* / down *Chart*s;
- Os índices visuais sobre o estado do projeto são "aquecedores" de informações;
- A gestão de *stakeholders* é feita através do *Product Owner*;
- O custo e cronograma são previsíveis e estáveis;
- *Release*/ *Sprint Burndown Chart*s mostram o desempenho ao vivo das *features* do projeto.

Gestão de Riscos

- O planeamento de riscos faz parte do planeamento da *release* / *sprint* e de reuniões de revisão;
- Toda a equipa está envolvida no planeamento e mitigação de riscos;
- A monitorização é assumida como uma parte do planeamento e da revisão da equipa.

Gestão de Contratos

- A equipa fornece inputs para definir os requisitos de aquisição usando pré-iterações de conceitos;
- A equipa *Scrum* realiza evoluções e fornece inputs para documentos contratuais;
- *Scrum* permite a rescisão antecipada dos contratos, citando como tipo de contrato "Money for Nothing" - um cliente pode cancelar um contrato no final dos *sprints* pagando 20-30% do valor do contrato 'Change for Free';
- O *sprint* extra (*Sprint* N + 1) pode ser usado para tarefas de administração formal.

5 Trabalho desenvolvido

Este capítulo retrará a contribuição desta dissertação para a comunidade científica.

A metodologia ágil reconhece que os requisitos do projeto poderão ser alterados; os pedidos de mudança podem ser incorporados nas futuras iterações em questão de semanas. O projeto ágil pode acomodar essas mudanças devido ao seu ciclo de *release* curto (*sprint*).

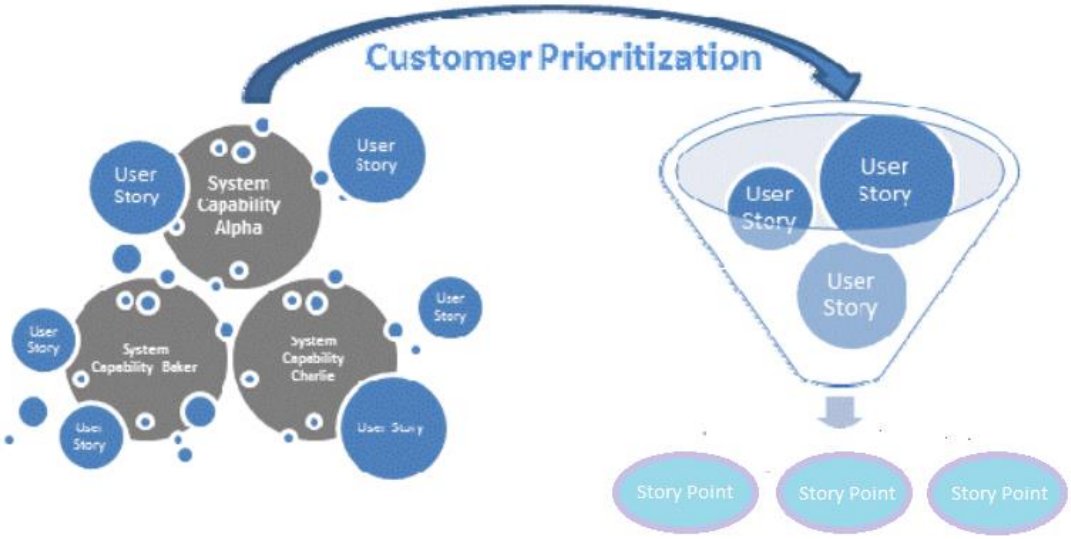

Figura 6 - Carga de trabalho para um Sprint adaptado de (Hayes et al., 2014)

No projeto ágil, o produto é dividido em *features*, que são atribuídos a múltiplos *User Story*s. As *features* são então designadas por *User Story*s que por sua vez são divididas em tarefas (*Story Points*). As iterações (*sprint*) são baseadas nas prioridades das *User Story*s. Análise, *design*, codificação, teste e implementação são feitos apenas para as *User Story*s que estão na iteração atual.

Earned Value Management (EVM) é uma metodologia de gestão para monitorar e controlar o tempo, custo e âmbito. Permite à gestão medir e acompanhar a execução e o progresso do projeto (Glaiel, 2012).

Os custos podem ser adicionados aos gráficos *Sprint Burndown* e *Release Burn up* para visualizar a informação juntamente com a taxa de conclusão de *features*. Tudo isso pode ser mais valioso para o gestor do projeto e para os *stakeholders* do projeto na monitorização de um projeto ágil, ao invés de tentar aplicar o *Earned Value* tradicional (Cabri & Griffiths, 2006).

O cálculo EVM no projeto ágil será baseado na Iteração (*sprint*). As mudanças do âmbito geralmente são implementadas no futuro ou futuras iterações. Como tal, o trabalho de iteração atual pode ser predefinido e não será alterado na duração do desenvolvimento do *sprint* (Roy & Goutam, 2014).

Atualmente no *Scrum* utilizam-se 3 métricas principais (Hayes et al., 2014):

- Velocidade;
- *Sprint* burndown *chart;*
- *Release burn up chart;*

A velocidade (uma métrica que varia consoante a equipa e experiência dos seus elementos) é a quantidade de *story points* completos por cada iteração (*sprint*) ao longo de cada *release*.

A velocidade permite que a gestão de TI com base nos ciclos de planeamento prévios melhore as estimativas da quantidade de trabalho que pode ser feito nos *sprints* subsequentes. O gráfico *Release Burndown* mostra uma imagem das tendências do projeto, que podem ser usadas para a previsão da data de conclusão. Ele reflete o impacto que uma adição (ou remoção) de *story points* tem na data de conclusão e na funcionalidade da *release*. O gráfico de *burndown* de *Sprint* mostra as tendências do projeto durante um *Sprint*, que pode ser usado para a previsão do cumprimento do âmbito até ao final do *Sprint* (Mahnic & Zabkar, 2012).

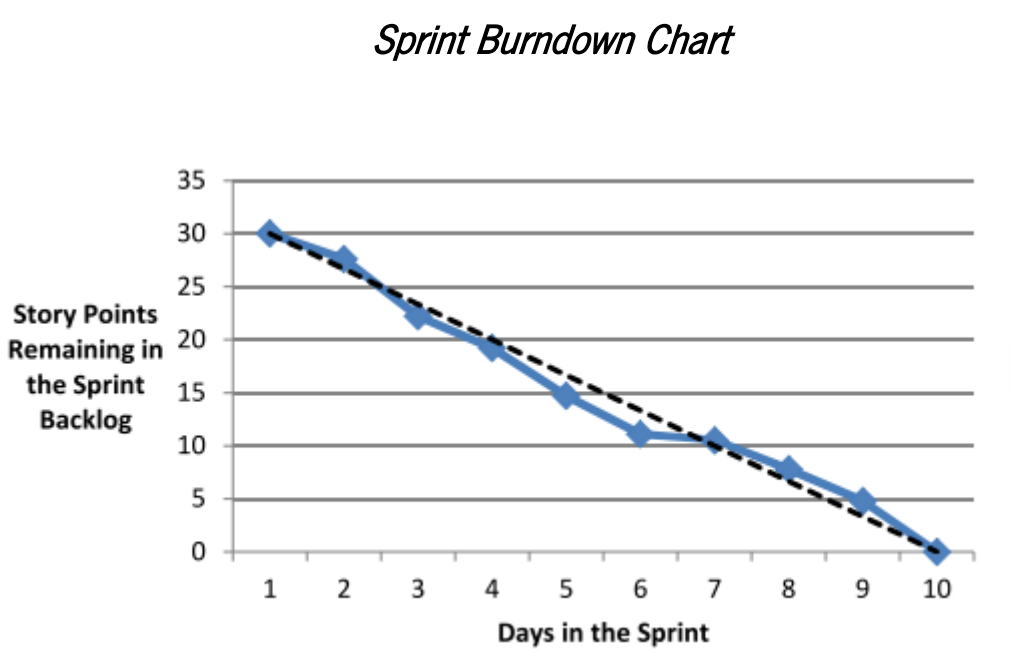

Figura 7 - Exemplo de Sprint Burndown Chart (Hayes et al., 2014)

A figura 7 é um *sprint burndown chart* que mostra as *story points* restantes e realizadas a cada dia do *sprint.* Neste gráfico podemos visualizar que a equipa manteve uma velocidade relativamente estável desde o primeiro dia de *sprint* até ao último e que cumpriu os trinta *story points* previstos para a realização do *sprint*.

Release Burn up Chart

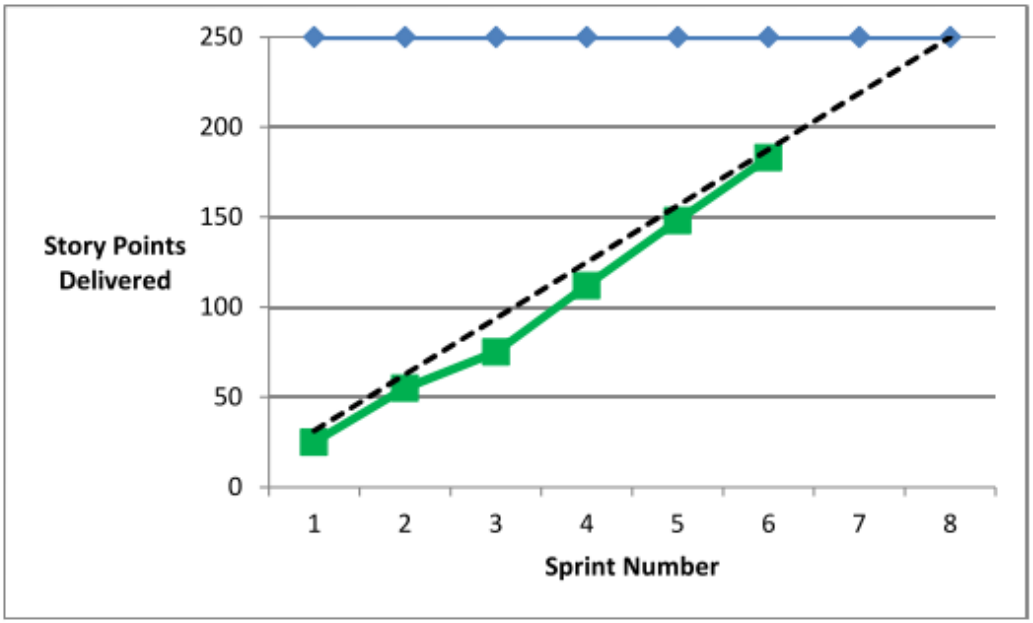

Figura 8 - Exemplo de Release Burn up Chart (Hayes et al., 2014)

A figura 8 mostra um exemplo de um gráfico *burn up*, onde se mostram os *story points* completos por cada *sprint* e a sua evolução global ao longo do projeto. Neste gráfico podemos visualizar que a equipa de desenvolvimento alcançou os *story points* planeados. No primeiro e segundo *sprint* a equipa quase conseguiu atingir a velocidade expectável, mas teve uma quebra no terceiro *sprint* que foi compensada nos sprints seguintes até ao sexto sprint onde a equipa alcançou o valor estimado de story points completos atingindo a velocidade ideal.

Na próxima secção irá ser apresentado o referencial proposto nesta dissertação com o nome de *Scrum*EVM visto combinar o método EVM com a metodologia ágil *Scrum*. Este referencial foi detalhado, explorado e refinado com:

- Brainstorming
- Pensamento criativo
- Pensamento racional

A partir da revisão de literatura e com o contributo de dois especialistas de gestão de projetos.

5.1 Proposta *Scrum*EVM

As métricas EVM que serão utilizadas no *Scrum*EVM são adaptadas do *Agile*EVM e são as seguintes:

Tabela 8 - Métricas de ScrumEVM adaptadas de AgileEVM (Sulaiman et al., 2006)

Nome	Definição
Planned *Value* (PV)	*Story Points* planeadas
Earned Value (EV)	*Story Points* completadas
Actual *Cost* (AC)	Custo real
Schedule Performance Index (SPI)	EV/PV
Cost Performance Index (CPI)	EV/AC

Sulaiman et al. (2006) adaptou o *Earned Value Management* (EVM) usando valores definidos em *Scrum*. O resultado é chamado *Agile*EVM (*Agile Earned Value Management*) e é um conjunto de cálculos simplificados de *Earned Value*.

*Agile*EVM realiza um ajuste mental com foco no planeamento, na entrega e nos relatórios sobre o valor obtido pelas capacidades (Park, 2010).

Técnicas de gestão de projetos ágeis, como gráficos de *burn up* (indicando a quantidade de funcionalidades pendentes vs. completas ao longo do tempo, etc.) fornecem informações de status e progresso muito semelhantes ao que o *Earned value* tenta medir (Cabri & Griffiths, 2006).

Propõe-se a inserção de métricas do EVM no *Scrum*EVM (Tabela 8)Tabela 8 - Métricas de ScrumEVM adaptadas de AgileEVM *(Sulaiman et al., 2006)*, ao nível do *sprint* (iteração) e ao nível da *release*. A figura 9 procura representar os contributos desta proposta incorporados na metodologia ágil *Scrum* assinalados a cor vermelha.

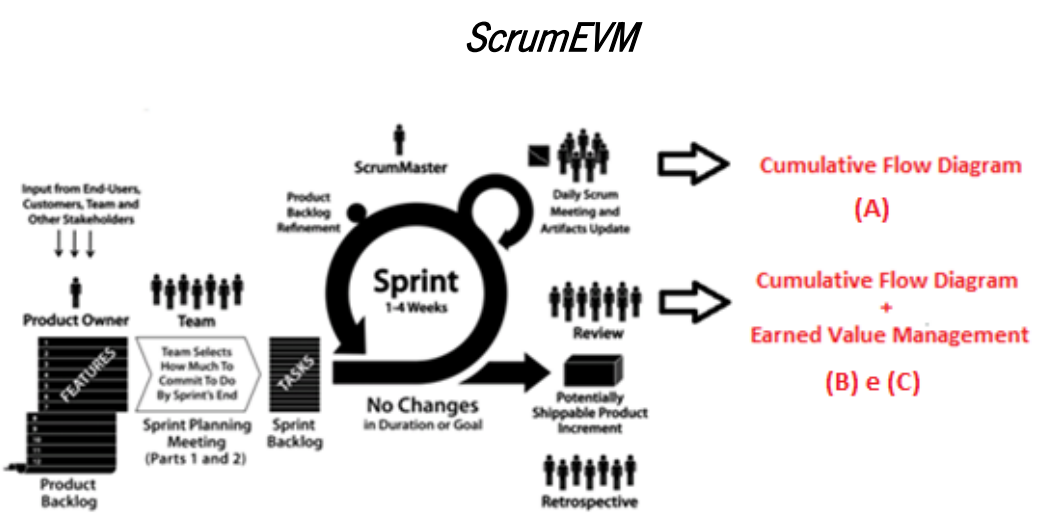

Figura 9 – Referencial ScrumEVM adaptado de (Deemer et al., 2012)

De seguida detalha-se a implementação do *Scrum*EVM.

A figura 9 apresenta o referencial *Scrum*EVM onde estão representados os papéis, eventos e artefactos bem como alguns *input*s e *output*s.

Propõe-se a inserção de um Cumulative Flow *Diagram* (CFD), conceito importado de Lean *Product* Development (Reinertsen, 2009), na Daily *Scrum* Meeting bem como um CFD no *Sprint Review* complementado por gráficos com métricas EVM.

O *Scrum*EVM em nada altera ou excluí artefactos ou eventos da aplicação normal do *Scrum*. Acrescenta gráficos com informações que podem ser relevantes para todos os *stakeholders* do projeto. A informação retirada pelas métricas EVM concentra-se no nível orçamental, enquanto os C*umulative Flow Diagram* terão o papel de acrescentar detalhe relativamente ao trabalho que está em progresso durante a realização de cada *Sprint* bem como fornecer informação mais detalhada que permita uma análise mais rigorosa de cada *Sprint* realizado.

Esta técnica fornece um meio eficaz para medir o desempenho de uma variedade de perspetivas e está a ter um maior uso entre os desenvolvedores *Agile* (Hayes et al., 2014). A técnica assenta o foco em itens de trabalho que passam por um conjunto de transições de estado e identifica eventuais estrangulamentos e de risco no processo.

Os diferentes estados do CFD são os seguintes:

- *Ready*: A equipa analisou a *user story* para determinar o seu tamanho e para procurar os esclarecimentos ou elaborações necessárias e a *story* está pronta para ser selecionada e inserida no próximo *sprint*.
- *Coding*: Depois de identificada como uma das *story*s de maior valor no *backlog*, a equipa agora está a trabalhar para a implementar.
- *Testing*: Depois de completado todo o trabalho de desenvolvimento, o código desenvolvido pela equipa está pronto para ser testado.
- *Done*: Depois do código desenvolvido ter passado com sucesso todos os testes, este código está potencialmente pronto para enviar.

Teve-se a preocupação de utilizar métricas com uma visualização gráfica simples para uma melhor e mais rápida compreensão das informações fornecidas, permitindo tirar conclusões da sua visualização e avaliação. Pretende-se que possam ser implementadas no *Scrum* sem acrescer qualquer tipo de eventos ou tarefas que afetariam negativamente a velocidade de cada equipa de trabalho.

O principal objetivo do *Scrum*EVM é o fornecimento dessas informações ao *Scrum Master*, para melhorar o seu trabalho de "Gestor de Projeto" conjugando um detalhe mais específico do trabalho em progresso

bem como perspetivas futuras ao nível orçamental, fazendo com que ele preveja possíveis problemas e os possa antecipar e corrigir a tempo. Com essas informações o *Scrum Master* pode interagir com os diferentes *stakeholders* do projeto e poder informá-los dos seus respetivos interesses e preocupações, seja ao nível do âmbito, orçamento, produto e até do desenvolvimento.

Adicionalmente, segundo Roy & Goutam (2014) o conceito de EVM é aplicável no projeto de *software* ágil, dado que as mudanças do âmbito geralmente são implementadas no futuro ou futuras iterações; como tal, o trabalho da iteração atual pode ser predefinido e não será alterado ao longo da duração do desenvolvimento do *Sprint*.

Assim, a implementação de *Agile*EVM concentra-se na medida do progresso ao nível da *release*, que segundo Roy & Goutam (2014) será a maneira mais apropriada de usar métricas de gestão de *earned value* em projetos ágeis.
Comparando com os requisitos do EVM tradicional, o *Agile*EVM alavanca itens de trabalho que são parte integrante do processo *Scrum*. O progresso é medido no final de cada *sprint* quando a velocidade real de *sprint* e os custos reais são conhecidos (Sulaiman et al., 2006).

A proposta de Torrecilla-Salinas et al. (2015) baseia-se na de Sulaiman et al. (2006), com algumas variações: primeiro, as métricas de EVM são obtidas ao nível do *Sprint* para fornecer informações de gestão valiosas com mais frequência. Em segundo lugar, não usam a abordagem *Agile* EVM para calcular a data da *Release*, mas sim para monitorar e controlar o projeto de modo a mantê-lo sob as suas restrições; e, por fim, mantêm a terminologia padrão do EVM.
A proposta de Roy & Goutam (2014) indica que as métricas EVM em projetos ágeis podem ser calculadas em cada iteração (*sprint*).
A análise que *Agile*EVM fornece, juntamente com o método de *burndown*, ajuda a substanciar a intuição e fornece aos executivos dados quantitativos de forma consistente (Sulaiman et al., 2006).
Os requisitos e a nossa compreensão sobre eles evoluem durante a vida do projeto. O *Agile*EVM aborda esta preocupação bem considerando a linha de tempo de uma única *release* (composta por vários *sprints*).
As estimativas do *Story Points* e a prioridade relativa das *story*s não são suscetíveis de mudar muito durante o período de uma única *release* (Hayes et al., 2014).

Uma advertência importante é que a mudança é esperada em projetos *Agile* e, portanto, as métricas *Agile*EVM são derivadas do que é verdadeiro em cada limite de *Sprint* (Sulaiman et al., 2006).

Propõem-se que a aplicação do *Scrum*EVM, ao nível do *sprint*, siga a metodologia detalhada de seguida:

(A) Na *Daily Scrum Meeting* serão apresentados os seguintes artefactos:

- *Cumulative Flow Diagram* (exemplo: figura 10);

- *Sprint Burndown Chart* (exemplo: figura 7).

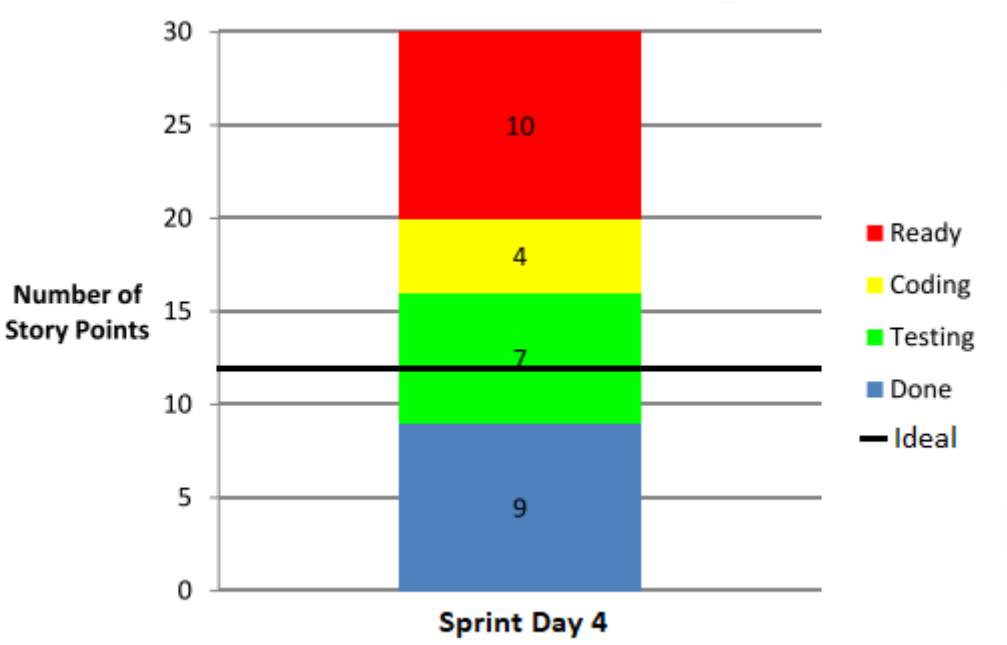

Figura 10 - Stacked Column Chart (adaptado de Hayes et al., 2014)

O gráfico da figura 10 é um exemplo de CFD com uma equipa de desenvolvimento *Scrum* com uma velocidade média de 3 *story points* por dia de iteração. Este gráfico indica que a equipa se encontra no quarto dia do seu *Sprint*. Quando o valor da velocidade é conhecido, é possível calcular o valor ideal de *story points* concluídas.

$$Ideal = Sprint\ Day\ x\ Velocity$$

Como a velocidade da equipa é de três *Story Points* por dia, o Ideal é de doze *Story Points* (4x3=12).

Podemos observar através deste gráfico exemplo que a equipa se encontra relativamente atrasada, sendo que apenas 9 *Story Points* estão completados, quando o ideal seriam 12. Mas podemos observar que 7 *story points* já se encontram na fase de testes e que outros 4 já se encontram a ser implementados. Com a visualização do trabalho em progresso normal do *Scrum* o trabalho que estaria supostamente atrasado (9/12) pode afinal estar até adiantado, caso a equipa de desenvolvimento consiga testar os 7 *story points* no próximo dia de trabalho.

Propõe-se que em tempo real, o gráfico iria ser atualizado à medida que a equipa de desenvolvimento completasse um *story point* ou este mudasse de estado. Com esta representação existe uma visualização mais realista dos *story points* ao invés da realidade do *Scrum* - tudo ou nada (0 ou 100%).

Com os gráficos CFD e *Sprint Burndown* atualizados à medida que o estado de cada *story point* seja alterado ou finalizado, cria um feedback instantaneamente positivo e garante que toda a equipa se preocupa em completar os *story points*.

(B) No *Sprint Review* serão apresentados os seguintes artefactos:

- *Cumulative Flow Diagram* (exemplo: figura 11);
- *Sprint Burndown Chart (exemplo:* figura 7)*;*
- *Release Burn up Chart (exemplo*: figura 8).

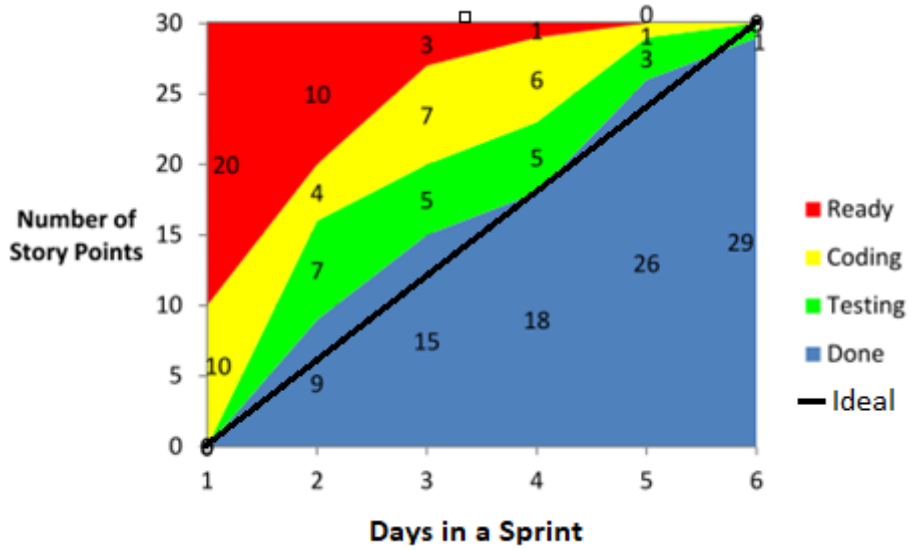

Figura 11 - Cumulative Flow Diagram during an iteration (adaptado de Hayes et al., 2014)

No exemplo representado na figura 11 a equipa encontra-se no final do *Sprint* onde podemos observar que apenas um *Story Point* não foi concluído, ainda se encontra na fase de testes.

Quando o trabalho em curso cresce substancialmente (*story points* na fase de *Coding* ou *Testing*), é um sinal de que a equipa pode estar presa com alguns *story points*, que permanecem inacabados. À medida que cresce o número de *story points* "presos", aumenta também o risco de que os desenvolvedores nunca tenham tempo para voltar atrás para os acabar (Hayes et al., 2014).

O CFD fornece uma compreensão mais detalhada do status do projeto em qualquer ponto no tempo, além de permitir a deteção precoce de problemas e consequentemente, a sua potencial correção.

Segundo Cabri & Griffiths (2006), os *Cumulative Flow Diagram*s (CFD) são outro método para medir o progresso num projeto de *software* ágil, com base nos gráficos *Burn up*. Tal como acontece com os gráficos de *burn up*, os CFDs retratam claramente o número proporcional de recursos concluídos ao longo do tempo. O benefício adicional dos CFDs deriva da sua representação do trabalho em progresso.

Um CFD fornece uma visão rápida e visual do trabalho em progresso geral num sistema e como esse trabalho está fluindo pelo sistema. Pode ser usado para calcular as trajetórias de *burn up* do trabalho, bem como para fornecer uma identificação visual rápida onde os estrangulamentos do processo estão a ser formados.

Os CFDs também podem mostrar os benefícios das métricas de *Earned value* da mesma forma que os gráficos de *burn up*. Além disso, o trabalho em progresso fornece uma indicação do tempo de execução, tamanho de iteração, ou seja, capturando o estado de transformação dá a uma estimativa para a realização das *features* que estão inseridas no trabalho em progresso. Estas são métricas úteis em projetos de *software*, e não estão disponíveis com o *Earned value* tradicional.

No final de cada *Sprint*, os *story points* que não estiverem implementados e/ou testados são aprovados automaticamente para o *sprint* seguinte, o que tem um impacto direto no planeamento feito de antemão.

(C) Ao nível da *release* propõe-se:

- a visualização de mais três artefactos, três gráficos com métricas do *Scrum*EVM com um conjunto de sprints durante uma Release (para uso exclusivo do *ScrumMaster*. (exemplos Figuras 12, 13 e 14);
- um Cumulative Flow *Diagram* com os *Sprints* acumulados durante uma Release (exemplo: figura 15).

Estes gráficos serão criados a partir do primeiro *Sprint* exclusive de cada *release* e terão um alcance desde o primeiro *sprint* até ao último sprint até à próxima *release*. O CFD com os *Sprints* acumulados (exemplo: Figura 15) será apresentado no *Sprint Review* a partir do segundo sprint até à próxima *release*.

O gráfico da figura 12 relaciona o *Planned Value* (PV) que é basicamente os *story points* previstos para cada *sprint* com o *Earned value* (EV), os *story points* com o estado *Done* (depois de serem implementados e passarem os testes). A diferença de este gráfico para o *Release Burn up Chart* é que em vez de *story points* no eixo vertical é traduzido por valor monetário.

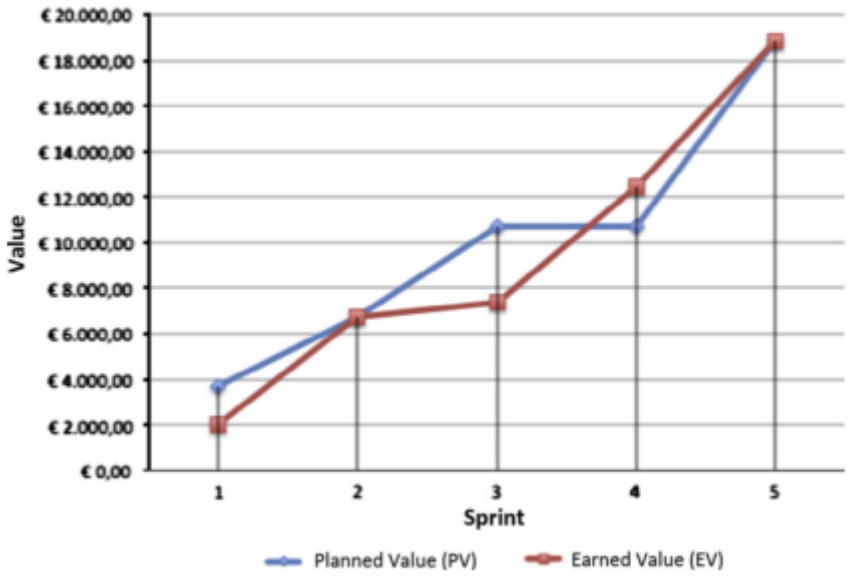

Figura 12 – PV e EV por Sprint durante uma Release (Torrecilla-Salinas et al., 2015)

No gráfico da figura 12 podemos visualizar que a equipa de desenvolvimento *Scrum* teve um pequeno atraso no primeiro *sprint* que foi compensado logo no *sprint* seguinte. Voltou a atrasar no terceiro *sprint*, esse atraso foi compensado no *sprint* quatro e até conseguiram implementar a mais que o previsto. No último *sprint* (*sprint* 5) o PV e o EV encontram-se sendo esta situação a ideal, visto que o planeado foi 100% cumprido.

O gráfico da figura 13 relaciona o *Earned value* (EV – *story points* com estado *done* no final de cada *Sprint*) com o *Actual cost* (AC – Custo real de cada *Sprint*).

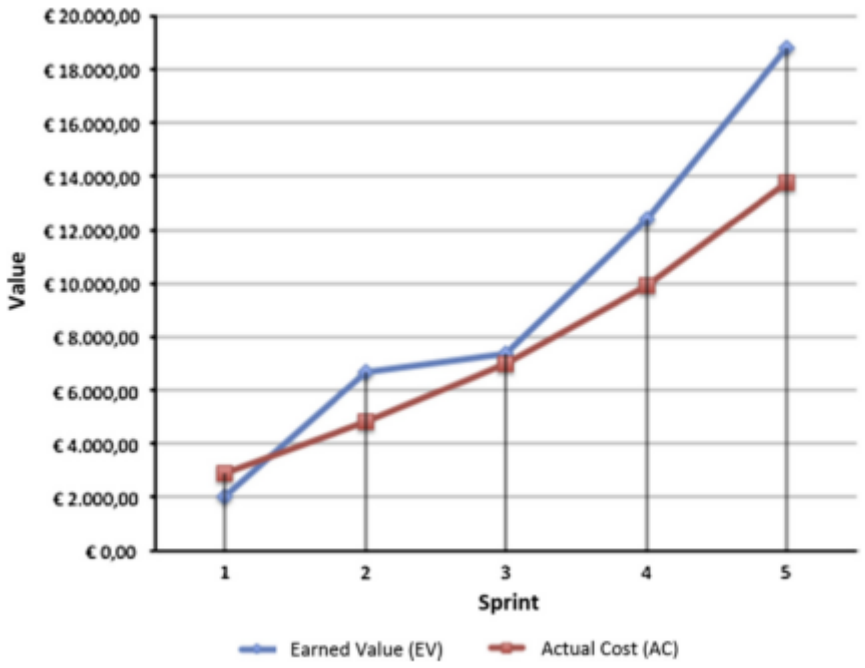

Figura 13 - EV e AC por Sprint durante uma Release (Torrecilla-Salinas et al., 2015)

No gráfico da figura 13, complementar ao anterior, conseguimos visualizar que apenas no primeiro *sprint* o AC é superior ao EV, no segundo *sprint* a equipa de desenvolvimento *Scrum* compensou, tendo apenas uma quebra de EV no terceiro *sprint*, mas apesar da quebra o EV continuava a ser superior ao AC. A partir do terceiro *sprint* a equipa conseguiu aumentar a margem entre o EV e o AC.

Através destas medidas podemos também calcular o SPI e o CPI que são métricas do EVM.

- SPI = EV / PV
- CPI = EV / AC

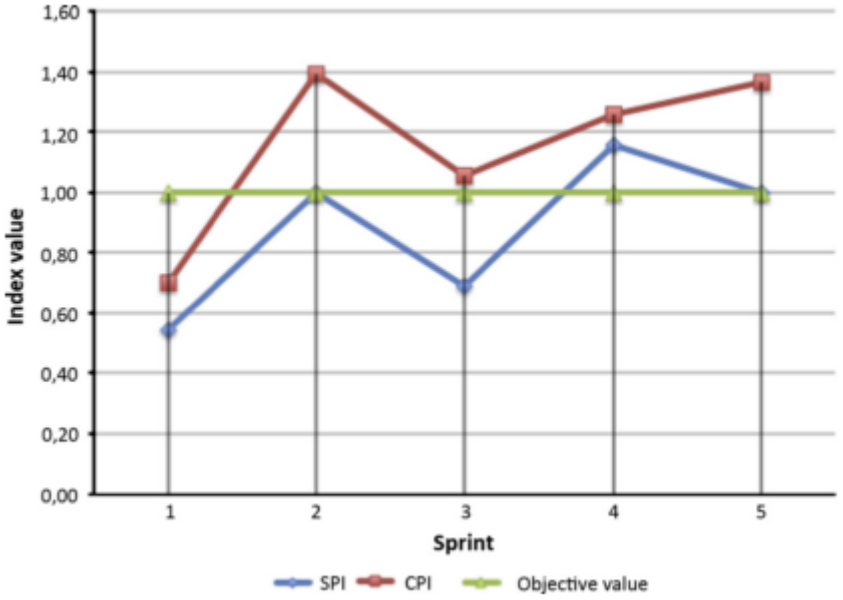

Figura 14 - Evolução do SPI e do CPI ao longo de cada Sprint (Torrecilla-Salinas et al., 2015)

No gráfico da figura 14 podemos visualizar o SPI e o CPI do projeto tendo por base o valor objetivo que é sempre igual a um. Como observamos nos gráficos anteriores, podemos também visualizar que no primeiro *sprint* o SPI e o CPI são inferiores a um. Podemos tirar a conclusão que o EV e o AC ficaram aquém dos objetivos planeados, verificando-se que no segundo *sprint* conseguiram compensar, atingindo os objetivos planeados de trabalho completado para o segundo *sprint* com um custo inferior ao planeado, havendo um decréscimo de trabalho realizado no terceiro *sprint* mas ainda com um resultado positivo no que toca ao nível orçamental. No quarto *sprint* conseguiram compensar novamente o trabalho realizado, apesar de terem diminuído a margem de lucro entre o EV e o AC mas ainda com um saldo positivo entre estes. No quinto *sprint* o PV é igualado ao EV finalizando o último *sprint* com o valor de SPI igual a um. A diferença entre o EV e o AC foi novamente alargado positivamente acabando com um saldo de CPI de 1.37.

Todas estas informações dão ao *Scrum Master* a informação necessária para conseguir fazer uma avaliação do projeto em andamento mais concreta e atual do que apenas com um *release burn up chart*, permite reportar aos *stakeholders* o estado do projeto com maior rigor e tomar medidas de prevenção, caso seja necessário.

O SPI apresenta informações semelhantes ao *burndown Sprint* na terminologia EVM, enquanto a CPI completa toda a imagem com a informação sobre a eficiência de custos, que não está incluída noutras medidas (Mahnic & Zabkar, 2012).

Uma medida equivalente ao SPI pode ser observada como: *story*s concluídas / *story*s planeadas. Quando as *story*s concluídas são menores do que as *story*s planeadas, a taxa de progressão é menor do que o planeado e o valor SPI é <1 (Cabri & Griffiths, 2006).

Nesse cenário, isso é uma indicação de que as estimativas de iteração precisam ser aprimoradas, e o âmbito do projeto possivelmente reajustado. Da mesma forma, uma medida equivalente ao CPI pode ser observada como: custos planeados / custos reais. Uma sobrecarga ocorre quando os custos reais excedem os custos planeados e o valor CPI é <1.

A grande vantagem da inclusão do EVM no *Scrum* é maioritariamente para os *stakeholders* do projeto terem uma visualização global do andamento do projeto, tanto dos *story points* com o tempo como dos *story points* com dinheiro, ao contrário do *Scrum* normal onde apenas se observa a evolução dos *story points* em tempo.

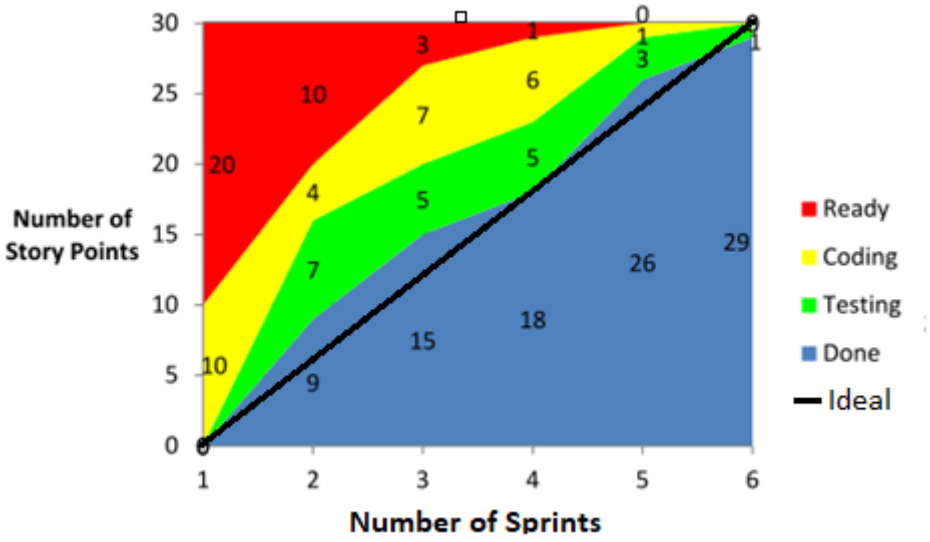

Figura 15 - Cumulative Flow Diagram durante uma Release (Hayes et al., 2014)

Neste exemplo de CFD (figura 15) podemos observar que uma equipa realizou 6 *Sprints* até uma *release*. Podemos concluir que a equipa de desenvolvimento *Scrum* teve um trabalho bastante positivo durante os primeiros *sprints* tendo atingido ou ultrapassado o valor ideal. Apenas no último e sexto *sprint* ficou um *story point* por testar. Ficou com uma taxa de *story points* concluídos superior a 96%. Cada *Sprint* é considerado com sucesso somente se um mínimo de 80% da Previsão Original for Aceite (Downey & Sutherland, 2013).

Todos estes artefactos (gráficos) serão atualizados à medida que cada *Sprint* é realizado, criando uma visualização global do projeto ao nível de cada *release*.

5.2 Resultados (da literatura)

Esta secção reporta resultados e conclusões de vários autores com casos práticos da aplicação do *Agile* EVM sendo o *Scrum* a metodologia ágil utilizada.

O trabalho de Roy & Goutam (2014) indica que o conceito de EVM é aplicável no projeto de *software* ágil. Embora a adoção do EVM seja limitada numa comunidade ágil, ela tornar-se-á popular com formação, educação e pesquisa adicional.

Segundo Cabri & Griffiths (2006) vários estudos de *Agile*-EVM foram apresentados, enfatizando a adoção de métricas ágeis existentes incorporando os conceitos de EVM, mantendo as práticas propostas de peso leve e de acordo com a Metodologia Ágil.

As equipas podem controlar as restrições do projeto, como horário ou custo, através de cálculos EVM com base na abordagem *Agile*. Estes cálculos, obtidos a nível de *Sprint*, podem ser apreciados como ferramentas úteis durante as retrospetivas de iteração, de modo a obter indicações sobre potenciais problemas nos processos de desenvolvimento. Isso pode aumentar o sentimento de abertura e o grau de compromisso das equipas e dos *stakeholders* ao longo do projeto (Torrecilla-Salinas et al., 2015).

Sulaiman et al. (2006) conclui depois de implementar o *Agile*EVM em dois projetos distintos de desenvolvimento de *software* que a implementação do processo *Agile*EVM não tem impacto notável na velocidade da equipa *Scrum*. Além disso, o valor dos dados foi confirmado pela equipa que teve acesso às métricas, bem como pelo *ScrumMaster* e pelos *stakeholders* de gestão do projeto. A análise de custos é valiosa para os *stakeholders* ágeis calculando o ROI estimado. Os *stakeholders* ágeis que são responsáveis por tomar decisões orçamentais consideram esta informação extremamente valiosa. Apetrechar a equipa e *stakeholders* ágeis com dados úteis e compreensíveis é vital para o "leme" com o qual a equipa *Scrum* é orientada, bem como para melhores processos e melhoria contínua. Ao fornecer as métricas *Burndown* e *Agile*EVM juntas, a equipa está melhor equipada para ter sucesso.

O estudo de caso de Mahnic & Zabkar (2012) evidenciou que cada medida adotada, Velocidade, *Release Burn up Chart*, *Sprint Burndown Chart*, SPI e CPI, é útil para medir o progresso dos projetos de *software* baseados em *Scrum* e que a recolha de dados não requer trabalho administrativo adicional que prejudicaria a agilidade do desenvolvimento. A velocidade permite que a gestão de TI seja aprendida dos ciclos de planeamento prévios e melhore as estimativas da quantidade de trabalho que pode ser feito nos *Sprints* subsequentes. O *Release Burn up Chart* mostra uma ampla imagem das tendências do projeto, que podem ser usadas para a previsão da data de conclusão. O *Sprint Burndown Chart* mostra as tendências do projeto durante um *Sprint*, que pode ser usado para a previsão do cumprimento do âmbito até ao final do *Sprint*. O SPI apresenta informações semelhantes ao *Sprint Burndown Chart* na terminologia EVM, enquanto a CPI completa toda a imagem com a informação sobre a eficiência de custos, que não está incluída nas outras medidas.

Segundo Kane (2007), o gestor de projetos no mundo ágil não é apenas o responsável pela tarefa, ou um profissional de planeamento, mas um líder que poderá visualizar e ver o horizonte. Ele gere o âmbito através

de mudanças. Enquanto isso, este gestor de projeto precisa de seguir os processos geralmente utilizados por projetos tradicionais, ou seja, iniciar o projeto, construir um plano e executá-lo, gerir o âmbito, estimar o custo, etc.

Projetos com ciclos de *Release*s extremamente baixos não poderão tirar tanto proveito da maior utilidade das técnicas de medição (como o EVM) como projetos com ciclos de *Release*s mais longos que envolvam vários *sprints* ou iterações.

Os resultados dos estudos reportados são aqui apresentados como garantes da mais-valia da minha proposta. Dada a impossibilidade física de testar a abordagem proposta em projetos reais, estes resultados poderão constituir uma validação da minha proposta para aplicação do método EVM no *Scrum*.

5.3 Boas Práticas

Esta secção irá conter um conjunto de boas práticas da aplicação da metodologia ágil *Scrum* bem como da aplicação do método EVM em *Scrum* sugeridos por vários autores e especialistas.

Rusk (2009) sugere:

- Ao apresentar o EVM aos *stakeholders* que não são gestores de projetos, considere apresentá-lo graficamente, sem siglas.
- As implementações simplificadas não precisam de custos reais por tarefa. As implementações simplificadas podem usar o valor planeado linear se estes usam técnicas ágeis para obter saída aproximadamente linear.
- Mostrar mudanças "ao vivo" para EV é bom. Isso aumenta a motivação se a equipa puder ver o trabalho do dia com impacto direto na linha de EV.

Uma boa métrica ágil ou diagnóstico, segundo Hartmann & Dymond (2006):

- Afirma e reforça os princípios *Lean* e *Agile;*
- Mede o *outcome*, não o *output;*
- Segue as tendências, não números;
- Responde a uma pergunta particular para uma pessoa real;
- Pertence a um pequeno conjunto de métricas e diagnósticos;
- É fácil de recolher;
- Revela o seu contexto e variáveis significativas;
- Fornece conteúdo para uma conversa significativa;
- Fornece comentários de forma frequente e regular;
- Pode medir o Valor (produto) ou Processo;
- Encoraja a qualidade "suficientemente boa".

Implementando a abordagem EVM simplificada, Rusk (2009) sugere ainda que:

- Simplificamos a distribuição do valor planeado ao longo do tempo. Podemos fazer isso porque os projetos ágeis são projetados para ter um resultado linear ao longo do tempo - produzindo aproximadamente a mesma quantidade de *output* em cada iteração;
- O gráfico é o nosso único formato para informar o valor obtido; então não fornecemos tabelas numéricas. O gráfico cobre todos os trabalhos atuais no projeto;
- Cada semana, simplesmente registamos o número de horas trabalhadas no projeto, durante essa semana, por cada membro da equipa. Então, somamos as horas e multiplicamos pela taxa horária, para obter o custo da semana.
- Não nos relacionamos com o nosso sistema financeiro ou de horários. Para economizar em custos de integração e configuração, cada membro da equipa simplesmente lê as horas totais da semana para o gestor do projeto. Isso funciona porque não precisamos de custos ao nível da tarefa, apenas as horas por semana.
- Temos uma "regra de ganhos" muito simples: o *earned value* de uma tarefa é acumulado quando a tarefa está concluída.
- O ponto-chave é que as mudanças acontecem imediatamente. Quando os desenvolvedores concluíram as tarefas, eles imediatamente veem a linha do EV a subir um pouco. Este feedback positivo instantâneo aumenta a motivação.
- O gráfico funciona melhor quando abrange um período de tempo que é grande o suficiente para se sentir como a "grande imagem", mas ainda suficientemente pequeno para que a equipa possa ver a linha subir no decorrer de um dia (bem-sucedido).

O tamanho relativo de cada *user story* torna-se uma fonte de informação usada pela equipa de desenvolvimento para permitir que o cliente considere sequências e agrupamentos alternativos para o desenvolvimento do *software*. Alternativas como Hayes et al. (2014) sugere:

- Implementar as *features* mais visíveis em cada capacidade, para que os utilizadores possam vê-los como protótipos a serem elaborados ou refinados;
- Escolher a capacidade mais importante e criá-la primeiro para que um sistema parcial possa ser colocado em campo para atender às necessidades urgentes e o restante pode ser adicionado de forma incremental;
- Implementar *features* que representem a infraestrutura necessária para criar outras *features*.

Teste de desempenho ágil (Figura 16 - Custo dos defeitos (Vani et al., 2014)) tem resultados positivos em relação a metodologias mais tradicionais (Vani et al., 2014):

- Sintonização do código: o ajuste do nível da unidade é feito cedo e, portanto, a otimização do código na fase posterior não é necessária.
- Rejeição da aplicação: reduz o risco de rejeição do aplicativo por causa de bloqueios da base de dados.
- Deteção anterior: reduz o esforço e a duração para o ajuste do desempenho e o custo da deteção de defeitos.
- Datas de *Release*: leva a mais confiança nas datas de *release*.
- Bugs de Desempenho: menor número de bugs de desempenho em fases subsequentes.
- Esforços de Teste de Desempenho: o script de desempenho pode ser recusado, economizando assim 60% dos esforços do ciclo.

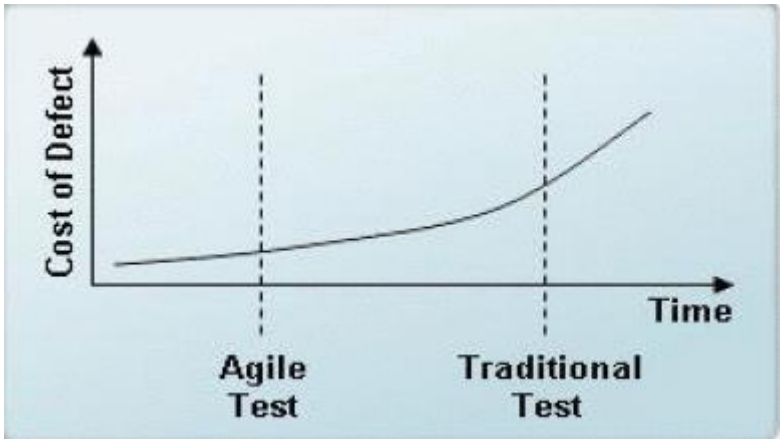

Figura 16 - Custo dos defeitos (Vani et al., 2014)

Hayes et al. (2014) definiu e detalhou oito categorias principais de métricas de medição de progresso para projetos ágeis:

- Tamanho do *software*

Quando se estabelece uma cadência constante de trabalho - com custo previsível – os objetivos mudam para maximizar a quantidade de funcionalidades com valor real para o cliente que são entregues. A partir desse ponto de vista, o tamanho torna-se um atributo variável e não um ponto de partida fixo. Usando a prioridade do cliente, a equipa esforça-se para equilibrar a carga selecionando as *story points* de maior prioridade que se encaixam na capacidade estabelecida. A associação de *story points* com *features* do sistema é uma maneira útil de fornecer contexto para interpretar *releases* - quando se considera uma perspetiva mais ampla do que o foco da equipa de desenvolvimento.

Ao medir o progresso da equipa, o foco está na sua capacidade de cumprir o compromisso de entregar o incremento, com software a funcionar. Portanto, a correspondência entre o número de *story points*

planeados e o número entregue torna-se o foco, e não apenas uma medida do que é entregue. O tamanho do *software* é utilizado aqui para ajudar a equipa a entender a sua capacidade. A métrica de tamanho serve como base para o diagnóstico do desempenho da equipa. Quando várias equipas de desenvolvimento trabalham para construir um sistema grande é importante entender que os *story points* geralmente são calibrados para equipas de desenvolvimento específicas e não para qualquer tipo de equipas de desenvolvimento.

- Esforço em mão de obra para desenvolvimento de software

O esforço em mão-de-obra tem interesse para estimar o pessoal a contratar ou adquirir já que é a principal componente do custo em contratos de desenvolvimento de *software*. No entanto, é raro que uma análise detalhada dos horários de esforço gastos seja o foco das métricas de medição do progresso - a menos que seja usado para diagnosticar deficiências de desempenho ou quando o contratante corre o risco de exceder o orçamento.

Com uma equipa totalmente integrada, os métodos ágeis alcançam maior velocidade, em parte porque o produto não transita no seu estado intermédio de um grupo de especialistas para outro. Nos modelos tradicionais, cada transferência aumenta a oportunidade de introduzir erros de ameaça à qualidade e à programação. Além disso, quando a equipa *Agile* incorpora todas as perspetivas que outros modelos tratam como papéis diferentes no processo, a interação e sinergia em tempo real entre essas perspetivas podem ser mais efetivas.

- Cronograma

A intenção dos métodos de desenvolvimento *Agile* é tratar custo e cronograma como parâmetros fixos e gerir o progresso de forma a maximizar a quantidade de funcionalidades de alta prioridade a serem entregues. Isso contrasta com as abordagens tradicionais, onde o cronograma e o custo são consequência das escolhas feitas no conteúdo e na qualidade do que é entregue. Portanto, as métricas tradicionais que se concentram em coisas como o desempenho do cronograma ou a divulgação do final de uma determinada atividade são menos significativas nas abordagens de desenvolvimento *Agile*.

- Qualidade e satisfação do cliente

O cliente normalmente desempenha um papel proeminente na priorização de *user story*s, além de fornecer feedback atempado na conclusão de cada *sprint*. Portanto, a visão de qualidade não é relegada para contar defeitos ou falhas na funcionalidade fornecida. As métricas que quantificam o valor do negócio para cada *user story* podem ser recolhidas - com base num número atribuído pelo cliente para cada *story* - embora isso seja normalmente ignorado em favor do cliente que participa como árbitro nas prioridades para o conteúdo de cada iteração e / ou cada *release*.

A escolha de usar métodos *Agile* não anula o papel tradicional de atividades como Testes de Sistema, Testes de Qualificação Formal ou Teste de Aceitação de Cliente. Os defeitos descobertos e corrigidos durante essas atividades de teste fornecem uma base para métricas da mesma maneira que as mesmas fazem em abordagens de desenvolvimento mais tradicionais. Os ciclos de feedback especificamente projetados no processo servem para focar a equipa de desenvolvimento na qualidade como é visto pelo cliente.

- Custo e financiamento

O uso de métodos ágeis não requer uma mudança nos mecanismos de recolha associados ao custo do trabalho realizado - embora tais métricas certamente não sejam um foco da maioria dos métodos ágeis.

- Requisitos

A consideração das métricas de análise de requisitos decorre das variações na forma como as organizações adotam métodos *Agile*. Em alguns ambientes, os requisitos no *backlog* são essencialmente listas de *features* e não contêm todos os detalhes necessários usados pelos desenvolvedores para direcionar as decisões de implementação. Essa informação é adquirida através de conversação direta com utilizadores ou seus substitutos.

- Entrega e progresso

Os métodos ágeis colocam uma maior ênfase nos produtos entregues, em vez de atributos do processo usado para criar esses produtos. Medir durações, custos e desempenho de cronograma, embora sejam importantes, tendem a receber menos atenção nas abordagens *Agile*. As contagens de *story points* entregues (incorporados no *software* a funcionar) são o bloco de construção mais proeminente nas métricas *Agile*.

- Sistema ágil de *Earned Value*

O uso de *Earned Value Management Systems* (EVMS) na medição dos principais esforços de desenvolvimento de gestão de projetos de *software* está bem estabelecido. Muitos profissionais de aquisição têm abordagens testadas no tempo para diagnosticar a saúde dos programas que supervisionam que dependem do EVMS. Sulaiman, Barton e Blackburn (2006) explicam como o EVMS pode ser empregue usando métricas ágeis.

Os principais processos de execução do *Scrum* segundo Wan et al. (2013) são:

Fase determinante:

- Desenvolver os requisitos reais dos clientes;
- Escrever uma descrição de projeto de uma página;

- Recodificar os requisitos do(s) cliente(s);
- Obter a autorização dos gestores seniores para executar o projeto;
- Discutir como atender aos requisitos com os clientes.

Fase do Planeamento:

- Definir todo o trabalho do projeto;
- Estabelecer o cronograma inicial do projeto;
- Avaliar o tempo necessário para concluir o projeto;
- Analisar e ajustar o cronograma do projeto;
- Avaliar os recursos necessários para concluir o projeto;
- Escrever o plano de gestão de riscos;
- Avaliar todo o custo do projeto;
- Registar o plano do projeto;
- Classificar o trabalho por ordem cronológica;
- Obter permissão dos gestores seniores para iniciar o projeto.

Fase de Iniciação:

- Recrutar o Gestor do Projeto;
- Construir o processo de gestão de mudanças de âmbito;
- Recrutar os membros da equipa do projeto;
- Gerir a comunicação da equipa;
- Escrever o documento descritivo do projeto;
- Determinar o cronograma;
- Construir as regras de funcionamento da equipa;
- Escrever o *work* package.

Fase de supervisão e controlo:

- Construir o sistema de execução e de relatórios;
- Reportar o estado do projeto;
- Supervisionar a execução;
- Lidar com pedidos de mudança de âmbito;
- Supervisionar os riscos;
- Identificar e resolver os problemas.

Fase de Decisão e Início da Iteração:

- Definir o processo de tomada de decisão para a gestão de clientes;

- Os clientes devem estar totalmente envolvidos neste processo;
- O ambiente deve ser completamente aberto e honesto;
- A determinação deve basear-se no valor comercial esperado;
- A solução deve ser formada de acordo com o objetivo do projeto.

Fase de Finalização:

- Obter a confirmação do cliente;
- Preparar as entregas e instalações.
- Escrever o relatório de encerramento;
- Iniciar a auditoria da execução.

A Lei de Brook's

A adição de mão-de-obra a um projeto de software atrasado faz com que o projeto fique mais atrasado (Brooks, 1975).

Brook atribui esse fenômeno a dois fatores principais. Um deles é o fato de que novas pessoas no projeto demoram tempo para se tornar produtivas, pois há uma curva de aprendizagem associado à introdução de novos membros da equipa. Independentemente da experiência de um desenvolvedor e da profundidade da experiência técnica, geralmente há conhecimento específico do projeto que é improvável que seja conhecido por uma nova pessoa. Durante este tempo de "aceleração", a equipa é menos produtiva como um todo. Isto é devido não apenas à baixa produtividade inicial dos novos membros da equipa, mas também é devido ao tempo que os membros mais experientes da equipa devem dedicar aos novos membros, fazendo com que a equipa experiente seja menos produtiva. O segundo fator tem a ver com os custos de comunicação e coordenação que aumentam à medida que o tamanho da equipe cresce.

Segundo a Lei de Brook's é importante ter-se em conta e avaliar os primeiros dois/três *Sprints* da primeira *release* e tirar-se conclusões antecipadamente caso o projeto esteja razoavelmente atrasado. Os *story points* foram subestimados? A equipa de desenvolvimento não consegue atingir a velocidade necessária para o cumprimento do prazo estabelecido? Para que caso seja necessário contratar mais mão-de-obra, isto aconteça no inicio e não próximo do fim do projeto, onde resultaria (caso fosse no fim do projeto) num adiamento superior ao até anteriormente estimado e ainda, um aumento dos custos.

Rusk (2009) sugere boas iniciativas para o *Scrum Master* motivar mais a equipa com a atualização ao vivo da reta do *Earned Value,* bem como sugestões para o *Scrum Master* explicar métricas EVM sem siglas ou conceitos complicados, apenas métricas visuais. Aconselha a utilização de gráficos simples para *stakeholders* que não estejam familiarizados com os conceitos do EVM.

As categorias que Hayes et al. (2014) especificou são importantes e merecem um acompanhamento constante em todas as fases do projeto *Scrum*:

- Tamanho do Software
- Esforço em mão de obra para desenvolvimento de software
- Cronograma
- Qualidade e satisfação do cliente
- Custo e financiamento
- Requisitos
- Entrega e progresso

Qualquer *Scrum Master* deve analisar ao pormenor todos os processos (Wan et al. 2013) e boas práticas propostas pelo *Scrum* para ter sucesso nos projetos de desenvolvimento de *software*.

5.4 Validação do *Scrum*EVM

Design research refere-se ao processo de investigação para a criação de artefactos, com o objetivo de garantir disciplina, rigor e transparência à investigação, para que o conhecimento obtido, para além de tecnológico, seja, também, científico (Carvalho, Ferreira, Silva, & Ferreira, 2012).

Design science visa criar conhecimento sobre o processo de *design*, particularmente relevante para as tarefas de *design*, nomeadamente dos Sistemas de Informação. *Design* science apresenta-se como um corpo de conhecimento (Carvalho et al., 2012).

Admitindo estas duas abordagens, a investigação realizada encaixa-se na *Design Research* pois apresenta-se como um processo de investigação para a criação de um artefacto.

A *design research* pode ser descrita como uma forma de pesquisa que envolve o *design* de alguma criação humana ou artefacto. Esses artefactos são projetados com algum propósito, ou seja, eles visam responder a alguma necessidade humana, existente ou prevista (Carvalho, 2012).

Segundo Carvalho (2012) existem quatro critérios para a validação do *outcome* de *design research*

- Sucesso do artefacto

  O primeiro elemento de validade é que os novos artefactos devem ser bem-sucedidos. O sucesso pode ser estabelecido em termos de medidas como:

  - Utilidade
  - Eficiência
  - Eficácia

- Generalização

O segundo elemento de validade está relacionado com uma característica importante do conhecimento científico - generalização. O conhecimento científico é geral no sentido de que não está situado. Isso significa que a sua aplicabilidade não se restringe a situações específicas.

- Novidade

O terceiro elemento de validade considera novidade. A pesquisa produz novos conhecimentos. No caso da *design research*, há novos conhecimentos se houver um novo artefacto que corresponda a uma nova classe de artefactos, ou exiba melhorias significativas em relação aos artefactos existentes de alguma classe. Os pesquisadores do projeto devem demonstrar que o conhecimento que eles produzem é novo.

- Capacidade de explicação

O quarto e último elemento a considerar na validade dos resultados decorrentes da *design research* é que as razões para o sucesso dos objetos projetados devem ser explicadas. Apenas projetar um novo artefacto que seja bem-sucedido na realização do seu propósito não é suficiente. O investigador deve ser capaz de explicar por que é útil, eficaz ou mais eficiente do que artefactos alternativos. Isso implica que deve haver uma compreensão dos fenómenos que permitem a realização e desempenho de um artefacto e / ou dos fenómenos que englobam o seu uso ou operação.

Em relação ao primeiro elemento da validação de um artefacto que é a determinação do seu sucesso, ao nível da aplicação do *Scrum*EVM apenas será possível utilizar resultados práticos de outros autores que validaram previamente a aplicação das métricas EVM como é o caso de Sulaiman et al. (2006). A investigação destes autores concluiu, depois de implementar o *Agile*EVM em dois projetos distintos de desenvolvimento de *software*, que a implementação do processo *Agile*EVM, apesar de acrescentar métricas e nova informação, não tem impacto negativo na velocidade da equipa *Scrum*. Além disso, a utilidade ou valor dos novos dados foi confirmado pela equipa que teve acesso às métricas, bem como pelo *ScrumMaster* e pelos *stakeholders* de gestão do projeto. Vários outros estudos que fortalecem a validação segundo este elemento são apresentados nos resultados da abordagem proposta.

Em relação ao segundo elemento da validade de um artefacto, que é a generalização, considera-se que o *Scrum*EVM proposto satisfaz este elemento pois não tem qualquer tipo de restrição, podendo ser útil e aplicável por qualquer equipa de desenvolvimento *Scrum* para projetos na área de Sistemas de Informação.

Em relação ao terceiro elemento da validade de um artefacto, que é a novidade, o *Scrum*EVM satisfaz também este elemento pois é um artefacto que exibe melhorias em relação ao *Agile*EVM de Sulaiman et

al. (2006), acrescentando os Cumulative Flow *Diagram*s que detalham mais especificamente o trabalho em progresso.

Em relação ao quarto elemento da validade de um artefacto, a capacidade de explicação, considera-se também que este é cumprido, dado que o *Scrum*EVM proposto apresenta um racional lógico e detalhadamente explicado, implicando a existência da compreensão dos fenómenos que permitem a realização e desempenho do *Scrum*EVM, bem como a explicação da sua utilidade.

6 Conclusão e Investigação Futura

Neste último capítulo são retiradas as conclusões da realização da dissertação em termos teóricos e práticos e são ainda apresentadas propostas para investigação futura

6.1 Conclusão

No início da realização deste projeto foram definidas as atividades principais para a realização e concretização com sucesso desta dissertação de mestrado. De seguida foi elaborado um plano de atividades com um cruzamento das atividades mais importantes com as respetivas datas da sua realização.

Ao longo do desenvolvimento deste projeto de dissertação foi realizado um estudo extensivo sobre os conceitos e temas principais como o método EVM e os modelos ágeis. Depois de reunir conhecimentos mais sólidos sobre as demais temáticas foi realizado um estudo sobre a documentação fiável que abordava os dois mundos, os métodos mais tradicionais que utilizavam o EVM e os modelos ágeis.

Face à literatura consultada para o cruzamento das duas temáticas em análise perceciona-se que a investigação se justificava face à lacuna identificada e que era relevante preencher pela importância do cruzamento entre o método EVM com a modelo ágil *Scrum*.

Os projetos de desenvolvimento de *software* têm como objetivo uma entrega de um produto a um determinado cliente. É do interesse tanto do cliente como de todos os intervenientes que estão responsáveis pelo desenvolvimento e entrega do produto, que todo o processo abrangido pela sua concretização corra dentro do que foi estipulado, tanto no cronograma, âmbito, custo e requisitos. Ao longo dos anos as metodologias tradicionais de gestão de projetos como *waterfall* têm vindo a cair em desuso por inúmeras razões.

O *Earned Value Management* trata-se de um processo bem sistematizado de gestão de projetos, usado para encontrar variações nos projetos, com base na comparação do trabalho realmente feito face ao trabalho planeado. O EVM é utilizado para controlo de custos e prazos, podendo ser de grande utilidade para a previsão do projeto, sendo bastante utilizado em metodologias mais tradicionais de gestão de projetos de *software* como *waterfall*.

As metodologias ágeis têm vindo a ganhar terreno e a conquistar os especialistas em desenvolvimento e gestão de projetos de Tecnologias de Informação, por trazerem vantagens que as metodologias não ágeis não conseguem igualar com o mesmo esforço e tempo. Destaca-se em particular a superação que têm conseguido em tempo, custo e qualidade, comparando com metodologias mais tradicionais. As metodologias ágeis aceitam e abraçam a mudança de requisitos atualmente inerente aos projetos de desenvolvimento de *software*, e essa é a sua maior arma.

Dentro das diversas metodologias ágeis o *Scrum* é a metodologia favorita sendo conhecida e utilizada quase por 50% de todos os desenvolvedores ágeis (o *Scrum* está focado para entregar valor ao cliente no espaço de tempo mais curto possível). A metodologia *Scrum* desenvolve o produto numa série de iterações que estão inseridas num conjunto de *release*s que por sua vez compõe o produto final. A entrega frequente de *releases* ao cliente aumenta o valor do seu produto com o feedback contínuo do cliente. O *Scrum* ganha também com a sua política de testes de código, evitando problemas maiores com a deteção de erros numa fase mais preliminar.

O *Scrum* está apenas focado na entrega de valor para o cliente eliminando qualquer tipo de tarefa ou métrica que não acrescente esse valor, como por exemplo documentação detalhada. Sendo assim abre asas a algumas lacunas inerentes à sua pouca documentação. Uma dessas lacunas é o facto de não existirem métricas para a gestão de tempo e custos, foco do conceito *Earned Value Management*. Para conseguir compatibilizar EVM com *Scrum*, teve de abandonar-se a ideia de implementação do EVM tradicional no *Scrum* devido à mudança do âmbito que está associado a um projeto ágil. Deste modo é necessário fazer algumas alterações às métricas tradicionais do EVM bem como às suas conclusões.

Através da criação do conceito *Agile*EVM, documentado na literatura, foi possível combinar algumas métricas do EVM na metodologia ágil *Scrum* obtendo métricas importantes para o Gestor de Projetos, fornecendo perspetivas atuais do projeto ao nível de trabalho realizado com custos, com tempo e fazendo previsões para o futuro, algo que no *Scrum* não era normalmente utilizado.

Com a metodologia agora proposta, o *Scrum*EVM, para além de se obter as métricas *Agile*EVM existentes, consegue-se avaliar o trabalho em progresso de uma forma mais clara e detalhada. A combinação de métricas gráficas permite uma visualização e perceção das mesmas de forma rápida sendo este um ponto crítico para o *Scrum*, pois existe a possibilidade de o acréscimo de métricas começar a tornar a metodologia mais pesada perdendo a sua agilidade.

O objetivo final do *Scrum*EVM é fornecer ao Gestor de Projeto *Scrum* conhecido como *Scrum Master* a informação mais detalhada do trabalho em progresso que conjugado com as métricas EVM consegue dar uma perspetiva geral mais realista do estado presente da execução do projeto. Permite ainda ao Gestor do Projeto uma mais clara e atrativa ferramenta de diálogo com os *stakeholders* do projeto sempre que necessitar de introduzir alterações ao planeado ao longo do projeto. O *Scrum*EVM com a utilização gráfica das métricas EVM fornece previsões rápidas e realistas para futuras iterações e *release*s do produto.

O *Scrum*EVM proposto sendo uma combinação de duas metodologias genéricas é também por si genérico, ou seja, pode ser aplicado sem qualquer tipo de restrição contextual ou situacional, a qualquer projeto *Scrum* de desenvolvimento de *software*.

6.2 Investigação Futura

Depois do desenvolvimento deste trabalho de dissertação, como trabalho futuro sugere-se a implementação do *Scrum*EVM num projeto de desenvolvimento *software*, onde se pudesse avaliar o sucesso do artefacto com um estudo de caso ou vários se possível.

Como investigação futura sugere-se ainda abordar o defeito que é inerente a qualquer projeto de desenvolvimento de *software* e que seria interessante adicionar ao *Scrum*EVM alguma perspetiva ou abordagem de forma a evitar e prever defeitos de programação, aumentando assim a velocidade da equipa de desenvolvimento e a qualidade do produto final.

Referências

Abran, A., Moore, J. W., Dupuis, R., & Tripp, L. L. (2004). *Guide to the software engineering body of knowledge (swebok). 2004 ed P Bourque R Dupuis A Abran and JW Moore Eds IEEE Press*. https://doi.org/10.1234/12345678

Anbari, F. T. (2003). Earned Value Project Management Method and Extensions. *Project Management Journal*, *34*(4), 12–23. https://doi.org/10.1109/EMR.2004.25113

Bassil, Y. (2012). A Simulation Model for the Waterfall Software Development Life Cycle. *International Journal of Engineering & Technology*, *2*(5), 2049–3444. https://doi.org/10.15680/ijircce.2015.0305013

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. *XP Series*. https://doi.org/10.1136/adc.2005.076794

Beck, K. (2001). Aim, fire. *IEEE Software*, *18*(5), 87–89. https://doi.org/http://dx.doi.org.libezproxy.open.ac.uk/10.1109/52.951502

Burba, D. (2015). Still Proving Its Value. *PM Network*, *29*(2), 52–61. Retrieved from http://ezproxy.library.capella.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=100729861&site=ehost-live&scope=site

Cabri, A., & Griffiths, M. (2006). Earned value and Agile reporting. *Proceedings - AGILE Conference, 2006*, *2006*, 17–22. https://doi.org/10.1109/AGILE.2006.21

Carvalho, J. A. (2012). Validation Criteria for the Outcomes of Design Research

Carvalho, J. A., Ferreira, I., Silva, C., & Ferreira, S. (2012). Dilemas iniciais na investigação em TSI.

Cohen, N., & Gan, R. (2014). How to Manage an Agile / Kanban Software Project Using EVM. *International Journal of Computer and Information Technology*, *3*(3), 532–536.

Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2012). A Lightweight Guide to the Theory and Practice of Scrum, 1–20.

Despa, M. (2014). Comparative study on software development methodologies, *V*(3), 37–56.

Downey, S., & Sutherland, J. (2013). Scrum metrics for Hyperproductive Teams: How they fly like fighter aircraft. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 4870–4878. https://doi.org/10.1109/HICSS.2013.471

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9* (August), 28–35. https://doi.org/10.1177/004057368303900411

Glaiel, F. (2012). Agile Projects Dynamics: A Strategic Project Management Approach to the Study of Large-Scale Software Development Using System Dynamics. *Sloan School of Management*, *Master*, 137.

Gonçalves, C. (2016). Cláudia Alexandra Cruz Gonçalves Análise da aplicação do método EVM em projetos de TI.

Hall, N. G. (2012). Project management: Recent developments and research opportunities. *Journal of Systems Science and Systems Engineering*, *21*(2), 129–143. https://doi.org/10.1007/s11518-012-5190-5

Hartmann, D., & Dymond, R. (2006). Using Metrics and Diagnostics to Deliver Business Value, 2–7. https://doi.org/10.1109/AGILE.2006.17

Hastie, S., & Wojewoda, S. (2015). Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch. 1–9. Retrieved from http://www.infoq.com/articles/standish-chaos-2015

Hayes, W., Miller, S., Lapham, M. A., Wrubel, E., & Chick, T. (2014). Agile Metrics: Progress Monitoring of Agile Contractors. CMU/SEI-2013-TN-029, (January), 58.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly : Management Information Systems*, *28*(1), 75–106. https://doi.org/10.2307/25148625

Highsmith, J. (2002)a. *Agile Software Development Ecosystems*. *Solutions*. Retrieved from http://portal.acm.org/citation.cfm?id=513727

Highsmith J. (2002)b. What is agile software development? *Crosstalk*, *Highsmith*(Highsmith J. 2002. What is agile software development? Crosstalk, Highsmith, 4–9.), 4–9.

Ian Sommerville. (2010). In *Software Engineering (Ninth Edition)* (pp. 147–175).

Ilieva, S., Ivanov, P., & Stefanova, E. (2004). Analyses of an agile methodology implementation. *Proceedings. 30th Euromicro Conference, 2004.*, 1–8. https://doi.org/10.1109/EURMIC.2004.1333387

Kane, B. (2007). Estimating and Tracking Agile Projects. *PM World Today*, *IX*(V), 1–15. Retrieved from http://www.pmforum.org/library/studentpapers/2007/PDFs/Kane-5-07.pdf

Kantor, J., Long, K., Becla, J., Economou, F., Gelman, M., Juric, M., … Wu, X. (2016). Agile software development in an earned value world: A survival guide. *Proceedings of SPIE - The International Society for Optical Engineering*, *9911*, 1–18. https://doi.org/10.1117/12.2233380

Kerzner, H. (2009). *Project management: a systems approach to planning, scheduling, and controlling*. *New York*.

Mahalakshmi, M., & Sundararajan, M. (2013). Traditional SDLC Vs Scrum Methodology – A Comparative Study. *International Journal of Emerging Technology and Advanced Engineering*, *3*(6), 2–6.

Mahnic, V., & Zabkar, N. (2012). Measuring Progress of Scrum-based Software Projects, 73–76.

Manual, Q. (2006). Executive Summary. *Journal of the ICRU*, *6*(2), 7–8. https://doi.org/10.1093/jicru/ndl025

Munawar, H., & M., R. J. Q. (2015). Measuring the Effect of CMMI Quality Standard on Agile Scrum Model. *International Journal of Information Engineering and Electronic Business*, *7*(6), 46–52. https://doi.org/10.5815/ijieeb.2015.06.07

Oliveira, J., Vinhas, M., Costa, F., Nogueira, M., Ribeiro, P., & Machado, R. J. (2014). Is Scrum useful to mitigate the project risks in real business contexts ?

Palmquist, M. S., Lapham, M. A., Miller, S., Chick, T., & Ozkaya, I. (2013). Parallel Worlds: Agile and Waterfall Differences and Similarities. *SEI, Carnegie Mellon University*, (October), 1–101.

Park, J. M. (2010). Agile EVM 22 nd Annual Systems & Software Technology Conference, (April).

PMI. (2011). *A guide to the project management body of knowledge (PMBOK ® guide), fourth edition*.

PMI. (2013). *A Guide to the Project Management Body of Knowledge*. *Project Management Institute*

PMI. (2006). A Guide to the Project Management Body of Knowledge (PMBOK® Guide)-Third Edition. *PM Network*, *20*(3), 82.

Pressman, R. S. (2009). *Software Engineering A Practitioner's Approach 7th Ed* https://doi.org/10.1017/CBO9781107415324.004

Roy, P. K., & Goutam, P. (2014). Agile With EVM. *International Journal Of Core Engineering & Management*, *1*(8).

Rusk, J. (2009). Earned Value for Agile Development. *DoD Software Tech News*, *12*(1), 13. Retrieved from https://community.versionone.com/@api/deki/files/60/EarnedValueForAgileProjects.pdf

Seetharaman, B., & Mansor, Z. (2015). The development of agile cost management tool. *Proceedings - 5th International Conference on Electrical Engineering and Informatics: Bridging the Knowledge between Academic, Industry, and Community, ICEEI 2015*, 400–404. https://doi.org/10.1109/ICEEI.2015.7352534

Solomon, P. J. (2005). Performance-based earned value®. *15th Annual International Symposium of the International Council on Systems Engineering, INCOSE 2005*, *1*, 180–197. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84883307765&partnerID=tZOtx3y1

Sulaiman, T., Barton, B., & Blackburn, T. (2006). AgileEVM-earned value management in Scrum Projects. *Proceedings of the Agile Conference, Minneapolis (MN), USA, 23-28 July, 2006*, 10 pp.-pp.16. https://doi.org/10.1109/AGILE.2006.15

Torrecilla-Salinas, C. J., Sedeño, J., Escalona, M. J., & Mejías, M. (2015). Estimating, planning and managing Agile Web development projects under a value-based perspective. *Information and Software Technology*, *61*, 124–144. https://doi.org/10.1016/j.infsof.2015.01.006

Turley, F., & Office of Government Commerce. (2010). The PRINCE2 Training Manual-Practitioner Level. *MgmtPlaza*, 12 Introduction.

Usman, M., Soomro, T. R., & Brohi, M. N. (2014). Embedding Project Management into XP , Scrum and RUP, *10*(15), 293–307.

Vani, B., Suriya, B., & Deepalakshmi, R. (2014). Managing Performance of Web Based Application Through Agile Approach. *ICICES2014-S.A.Engineering College, Chennai, Tamil Nadu, India*, (978), 1–8.

Vasconcelos, B., Carvalho, D., Henrique, C., & Mello, P. (2011). Scrum agile product development method - literature review, analysis and classification Scrum agile product development method https://doi.org/10.4322/pmd.2011.005

VersionOne. (2006). Survey : " The State of Agile Development ," 5. Retrieved from http://www.versionone.com/pdf/2006-state-of-agile-survey.pdf

Wan, J., Zhu, Y., & Zeng, M. (2013). Case Study on Critical Success Factors of Running Scrum *, *2013*(February), 59–64.

Wu, S. (2012). Traditional and Agile Earned Value Management Processes. *Boston University*, 1–32.